ENERGY EFFICIENT PROCESSING IN MEMORY ARCHITECTURE FOR DEEP LEARNING COMPUTING ACCELERATION

A Dissertation
Presented to
The Academic Faculty

By

Yun Long

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2019

ENERGY EFFICIENT PROCESSING IN MEMORY ARCHITECTURE FOR DEEP LEARNING COMPUTING ACCELERATION

Approved by:

Dr. Saibal Mukhopadhyay, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Asif Islam Khan
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Hyesoon Kim
School of Computer Science
Georgia Institute of Technology

Dr. Tushar Krishna
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Shimeng Yu
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: August 21, 2019

ACKNOWLEDGEMENTS

First, I would like to especially thank my PhD advisor Dr. Saibal Mukhopadhyay, without his guidance and support I would not have the chance to work on these interesting topics, to learn how to do research. He has always been a great research advisor, mentor and source of knowledge. While offering me freedom to explore various exciting research topics and opportunities, he has been giving me a full-range of supports from forming the research ideas to establishing simulation/experiment environment, from research objective definition to paper writing. Besides providing valuable guidance and research vision, he is always generous to provide help for choosing the career path and all the aspects of life which I will remain grateful for all my life.

My sincere thanks also goes to the PhD defense committee members Dr. Asif Islam Khan, Dr. Hyesoon Kim, Dr. Tushar Krishna, Dr. Shimeng Yu, your suggestions and advises are very valuable and indispensable.

I also want to thank former and current Georgia Tech GREEN lab members, Dr. Amit Trivedi, Dr. Denny Lee, Dr. Wen Yueh, Dr. Zakir Khondker, Dr. Jaeha Kung, Dr. Duckhwan Kim, Dr. Monodeep Kar, Dr. Mohammad Faisal Amir, Dr. Taesik Na, Dr. Jong Hwan Ko, Dr. Arvind Singh, Burhan Mudassar, Edward Lee, Venkata Chaitanya Krishna Chekuri, Minah Lee, Nihar Dasari, Priyabrata Saha, Nikhil Chwala, Xueyuan She, Mukherjee Mandovi, Daehyun Kim, and Muhammad Arslan Ali. I would also like to extend my special thanks to Dr. Jaeha Kung, Dr. Taesik Na, Burhan Mudassar, Edward Lee, Xueyuan She, Daehyun Kim who have collaborated with me on several exiciting works. I wish all the best to their future research and other endeavors.

I also would like to thank Georgia Tech and ECE department for providing such a wonderful place to study and work.

I'm grateful to Mr. Poorna Kale and Micron Technology, for giving me the opportunity to work at the leading industry company and the chance to apply my knowledge to real

world applications.

At last, I want to say thank you to my family members, my beloved wife Shuping Kou and my parents, who give me unconditional supports and understanding all the time. I can not make it without you.

Many thanks to you all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The major objective of this research is to make the processing-in-memory (PIM) based deep learning accelerator more practical and more computing efficient. This research particularly focuses on the emerging non-volatile memory (NVM) based novel architecture design and leverages the software-hardware co-optimization to achieve the optimal computing efficiency without compromising the accuracy. From the emerging memory perspective, this research mainly explores resistive ram (ReRAM) and Ferroelectrical FET (FeFet). A dedicated recurrent neural network (RNN) accelerator is proposed which utilizes ReRAM as the basic computation cell for vector matrix multiplication (VMM). Regarding the challenges stemmed from ReRAM, this research also explores FeFET as an alternative solution. Dedicated data communication network, analog-digital conversion (ADC and DAC) free design are proposed to further enhance the efficiency. This research further proposes an flexible precision PIM design where the computation is performed with dynamical bit-precision. Besides the circuit and architecture optimization, algorithms are developed to fully utilize the hardware potentials. This research proposes a genetic algorithm (GA) based evolutionary method for layer-wise DNN quantization. DNN models can be dynamically quantized and deployed on the developed dynamic precision hardware platforms to achieve the best computing efficiency. To alleviate the accuracy drop caused by the device (such as ReRAM and FeFET) variation, this research proposes hardware noise aware training algorithm, leading to a reliable PIM engine with un-reliable device.

CHAPTER 1
INTRODUCTION

Deep neural networks (DNNs) have achieved unprecedented performance in solving complex problems in various domains, including computer vision [1, 2, 3], natural language processing [4, 5], robotics [6], autonomous driving [7], and gaming strategies [8, 9], to name a few. With the fast development of machine learning algorithms and the prosperity of deep learning, a big challenge is presented and becomes more critical than ever before: How to execute the machine learning computing fast and energy efficiently.

While most of the AI-related computing is still deployed on the conventional general-purpose processors (i.e. CPU and GPU) nowadays, extensive efforts have been spent to design dedicated hardware accelerators from both academia and industry. Application-specific integrated circuit (ASIC) and field-programmable gate array (FPGA) based solutions are the most matured and have been successfully deployed for real-world applications, ranging from cloud based server cluster, to embedded co-processor in mobile platform, to low-power chip used in internet-of-things (IoT) system [10, 11, 12, 13, 14, 15, 16, 17, 18]. While demonstrating better computing efficiency over general-purpose processors, the performance of ASIC and FPGA based designs are ultimately bonded by the computing density and data communication bandwidth. Several solutions (both hardware/architecture level optimizations as well as software/algorithms innovations) have been proposed to tackle these challenges.

From the software perspective, algorithms have been developed to reduce the DNN complexity (i.e. making the computing more efficient) while maintain the accuracy, including, quantization [19], pruning [19, 20], weight sharing [19, 21], knowledge distillation [22], etc. While most of these algorithmic approaches work decently well on shallow network and small dataset, they tend to be less effective when applying on deeper network

models and large dataset (such as ImageNet [23]).

From the hardware aspect, memory-rich architecture [10, 11] integrates large amount of on-chip memory to hold all the DNN model parameters and eliminates the off-chip dynamic random-access memory (DRAM) access, leading to fast and low cost data access, and thus, better efficiency. The near-memory-processing (NMP) architecture embeds logic engines within off-chip memory to reduce the cost of data-movement, [17, 18, 24]. While both techniques help to reduce the off-chip memory access latency and cost, the architecture is still bounded by logic-centric computation fabric where data and logic are separated. To further reduce the cost of data movement, a more aggressive approach is to directly perform computation inside memory, often referred to as the processing-in-memory (PIM) architectures [25, 26, 27, 28, 29, 30, 31]. The PIM designs truly eliminate the separation between memory and computing by re-purposing the memory array for computation, thereby realizing massive parallelism and almost nullifying data movement.

The PIM designs can successfully leverage emerging non-volatile memory devices, such as resistive ram (ReRAM), leading to non-volatile, high-density computing platforms [25, 26, 27, 31]. However, when examined closely from a circuit rather than micro-architecture perspective, we note that designing a scalable architecture with ReRAM based in-memory computation remains challenging and the high efficiency and high density advantages of ReRAM can not be fully utilized unless these challenges are properly addressed.

First, a crossbar with many parallel ReRAM devices presents a low-impedance resistive load. This is fundamentally at odds with CMOS gates which are designed to drive high-impedance loads (i.e. the gate of MOSFET). Power and area hungry analog drivers are necessary to ensure accurate computation in ReRAM crossbar. Second, as ReRAM has relatively low on-state resistance (1KΩ to 100KΩ for R_{on}) [32, 33, 34, 35], the energy dissipation during vector matrix multiplication (VMM) operation can be detrimental as all ReRAM devices in the crossbar simultaneously consume read current. Third, constrained

by the crossbar size as well as the system capacity, device re-programming are required to solve large problems. The high programming energy (>1 pJ/cell) [34, 33] in ReRAM can degrade the computing efficiency. Fourth, ReRAM is well known for its large device variation which inevitably introduce computing error, making ReRAM based PIM design less attractive for complex DNN model which typically very sensitive to parameter variation.

Besides these limitations which stemmed from ReRAM, there are several other obstacles that hinder the practical usage of PIM architecture. First, the analog-digital conversion (ADC/DAC) in prior works introduces large overhead for both power and chip area [25, 26]. Second, a well-developed software stack is still missing to bridge the gap before DNN model and its hardware deployment. Third, PIM itself can't efficiently perform computation beyond VMM (e.g. element-wise operation or transcendental functions used in recurrent neural network (RNN) [36]). Finally, most prior PIM designs employ a uniform bit-precision and lack the support for flexible bitwidth nor floating point precision which is the key factor for DNN training.

It is necessary and highly desired to develop a cross-cutting solution that integrates the full system stack ecosystem, the noise-robust DNN computing algorithms, architecture optimization, circuit re-design, and device level innovation to increase the feasibility of practical usage of PIM architecture and fully utilization the boundless potentials of the integration of the computing and the memory.

Towards this end, the major objective of this research is to make the PIM based deep learning accelerator more practical and more computing efficient. The research first proposes an ReRAM based RNN accelerator which integrates ReRAM based processing engine for VMM computing and digital logic for element-wise operation/transcendental function together, without compromising the versatility to support the computation for other types of DNN models. Regarding the challenges stemmed from ReRAM, this research explores Ferroelectrical FET (FeFET) to replace ReRAM as the basic memory cell in PIM architecture. A dedicated data communication network, named hierarchical network-on-

chip (H-NoC), is also presented in this work to enhance the data transmission efficiency. Moreover, to enable training and further enhance the computing efficiency, this research proposes an all-digital, flexible precision PIM design where the ADC/DAC are eliminated and the computation is performed with dynamical bit-precision. Fianlly, this research also proposes algorithmic approaches to suppress the computing accuracy deterioration caused by device variation, leading to a robust PIM engine with un-robust devices.

The rest of the thesis is organized as follows: In chapter 2, the detailed background and literature survey are presented. Chapter 3 presents ReRAM based PIM architecture for RNN computing acceleration. Chapter 4 presents the study of FeFET based PIM design and compared with the baseline ReRAM approach. Chapter 5 introduces FlexPIM, the core architecture to realize digital and flexible precision PIM design, as well as the algorithms for layer-wise DNN quantization. Chapter 6 discussed the algorithms developed to realize robust DNN computing with unreliable ReRAM device. Finally, chapter 7 describes the key research contributions and innovations, future research direction is also discussed.

CHAPTER 2
BACKGROUND

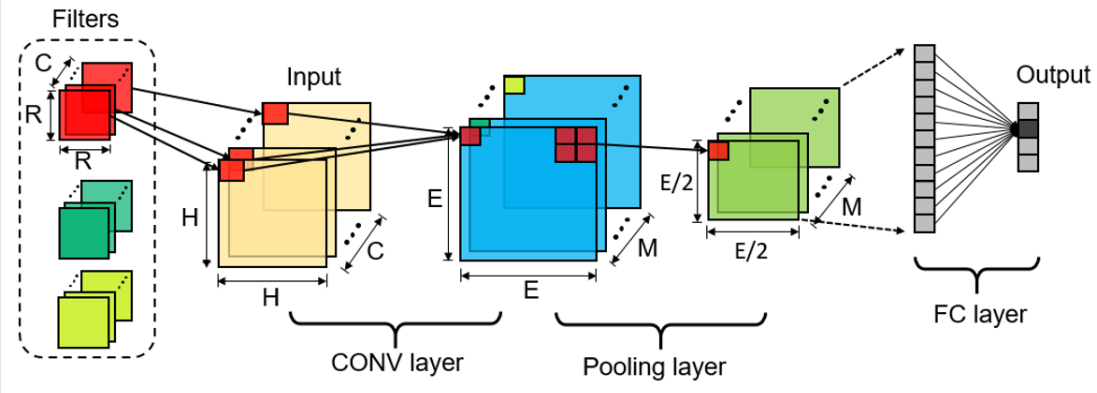
This section discusses the fundamentals for the proposed research work. First, the basic concept of convolutional neural network (CNN), recurrent neural network (RNN), and their computation is presented. Then the software and hardware approaches for accelerating DNN computing are discussed. The fundamental for processing-in-memory (PIM) is reviewed and candidate memory solutions are briefly discussed.

2.1 The computation of Deep Neural Network

2.1.1 Convolutional Neural Network

Benefiting from its powerful feature extraction capability, CNN is the corner stone for modern deep learning. Nowadays, most DNN models contain convolution layer to achieve the best performance across various application domains, including image processing [1, 2, 3], natural language processing [4, 5], robotics [6, 37], and autonomous driving [7], gaming AI [8, 9], etc.

A Convolutional Neural Networks (CNN) is composed of multiple layers including convolutional (Conv) layer, pooling layer and fully-connected (FC) layer, as shown in the top of Figure 2.1. The primary computation of CNN is the convolution operation where a set of filters are applied on the input feature maps (*ifmaps*), and generate the output feature maps (*ofmaps*). Pooling layer is utilized to down-sample the ofmaps to progressively reduce the spatial size of the representation and number of parameters. Activation function (ReLU, sigmoid, or other functions) provides more non-linearity to the networks. FC layer is similar with CONV layer, except that the filters have the same size as the ifmaps. Due to the nature of convolution operation (i.e. huge amount of matrices multiplication over the



DNN	AlexNet	CifarNet	LSTM	LeNet	ResNet	SVHN	VGG
% of VMM	99.8	99.8	99.9	99.4	99.9	99.9	99.5

*VMM: **V**ector **M**atrix **M**ultiplication

Figure 2.1: A typical CNN structure and the computation for CNN.

same weight kernel), the major computation for a CNN is about vector-matrix multiplication (VMM). While VMM operation dominates the computation, other type of operations, such as activation functions (ReLU or more complex transcendental functions) need be properly handled.

2.1.2 Recurrent Neural Network

Distinguished from CNN where data is feedforward in one direction, RNN takes both the current sample (x_t) and the previous calculated network state (h_t) as the network input (Figure 2.2). The feedback loop gives RNN the ability to remember and make decision based on previous information. In theory, RNN is capable of handling long-term dependencies with the feedback loop. However, in practice, it is difficult for RNN to have long-term memory due to the vanishing gradient problem [38].

LSTM are explicitly designed to combat vanishing gradients through a gating mechanism [39]. As shown in Figure 2.2, a LSTM cell contains three gates: input gate i_t, forget gate f_t, and output gate o_t. They are called gates because the sigmoid function squashes the values of these vectors between 0 and 1, and by multiplying them element-wisely with

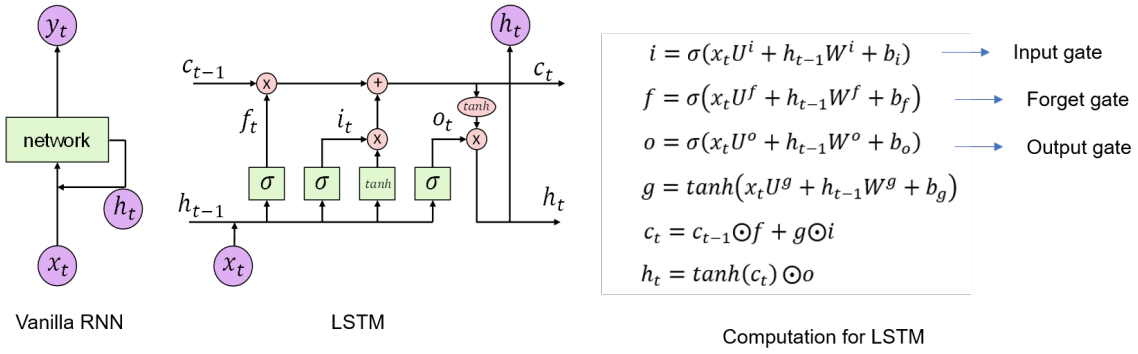


Figure 2.2: RNN and LSTM structure; LSTM computation definition.

another vector, we can then define how much information can pass through (i.e. behave like a gate). To be more specific, the forget gate defines how much of the previous state you want to let through; the input gate defines how much of the newly computed state for the current input you want to let through; and the output gate defines how much of the internal state you want to pass through. All the gates have the same dimensions which equal to the size of hidden state. The computation of LSTM is defined on the right of Figure 2.2. There are three main operation types: matrix-vector multiplication, non-linear activation function, and element-wise multiplication.

A variation of LSTM is called Gated recurrent unit, or GRU [40]. It combines the forget gate and input gate into a single update gate. The structure of GRU is simpler than the standard LSTM model, yet has demonstrated very good performance across various applications and been growing increasingly popular.

2.2 Software approaches for DNN computing acceleration

2.2.1 Quantization and Binarization

Low precision computing (quantization and binarization of DNN models) simplifies the hardware implementation, reducing the computing complexity and memory footprint. For example, 8-bit fixed point multiplication (which is used in the first generation of Google TPU [12]) saves 19.1x power than the 32-bit floating point multiplication operation [16].

Additionally, fetching same amount of data from on-chip SRAM can save 130x energy over accessing data from off-chip DRAM.

The early works on DNN quantization mainly focus on the inference pipeline, such as binary connect (BC) [41], ternary-weight network (TWN) [42], and incremental network quantization (INQ) [43]. While decent results are observed in these work, their model tends to be less accurate when applying to deeper DNN models and larger dataset.

More recently, quantization algorithms targeting the training pipeline are actively researched. XNOR-NET [21] proposes a DNN binarization algorithm where all the computation can be accomplished with simple XNOR operation. With AlexNet on ImageNet dataset, 16% accuracy loss is observed, which already outperforms the previous mentioned works [41, 42]. DoReFa [44] proposes to quantize both the DNN weight and gradient, achieving only 8% accuracy loss with the same evaluation metrics. Han et.al proposes Deep-compression [19] which integrates quantization, pruning (refer to 2.2.2), and Huffman coding to reduce the DNN complexity. With 4-bit weight precision, the accuracy drop is insignificant (less than 2%). The most recent work, Apprentice [22] which combines the quantization algorithm with knowledge distillation (refer to 2.2.3) together achieves the state-of-the-art performance. With 8-bit activation and 4-bit weight, the accuracy drop is less than 4% for ResNet-50 evaluated on ImageNet dataset.

2.2.2 Weight pruning

Weight pruning is another popular techniques which leverages the inherent redundancy in the number of weights in DNN models. A pioneering work of weight pruning is Deep-compression [19] which reduce the parameter size by 9x for AlexNet. During training, weight with small magnitude is pruned iteratively based on a threshold. The pruning algorithms are further optimized in several works, including Nest [45] which integrates pruning and auto-generation for DNN model design, and [46] which employs a dynamic network surgery approach to reduce network complexity by making on-the-fly connection pruning.

Heuristically, weight pruning is more effective than quantization [20]. First, there is often higher degree of redundancy in terms of number of parameters than the numerical bitwidth. Second, weight pruning is a type of regularization that enhance the salient weight while prune the unimportant ones, reducing the over-fitting effect. On the other hand, the major challenge of weight pruning resides in its hardware implementation due to its irregular sparsity and weight parameter indexing [20].

2.2.3 Other techniques

There are several other types of DNN reduction techniques.

Knowledge distillation. The general technique in knowledge distillation based methods involves using a teacher-student strategy. A deeper (more complex and larger) network is first trained. Then it is used to tech a smaller student network on the same task [22].

Hashing and weight sharing. A hash function is used to alias several weight parameters into few hash buckets, effectively lowering the parameter memory footprint [47].

2.3 Hardware approaches for DNN computing acceleration

2.3.1 ASIC and FPGA based DNN accelerator

The high demand for energy efficient execution of deep neural networks have motivated the fast development of DNN accelerators across various platforms including GPU [48, 49], ASIC [10, 16, 12], FPGA [15], and NVM [25, 26, 27, 29, 30, 50]. ASIC based designs employ specialized data path and highly optimized processing engine (PE), thereby, demonstrating better performance than the CPU/GPU baseline. For example, DaDianNao integrates large amount of on-chip eDRAM banks, realizing a near data processing framework [10]; EIE utilizes the weight compression algorithm to reduce the DNN parameter size, making it possible to fit large DNNs in on-chip memory, and thus, reducing the memory access energy [16]; and TPU from Google employs 65536 8-bit matrix multiply units using systolic execution, demonstrating $>$10x improvement on throughput/Watt over GPU

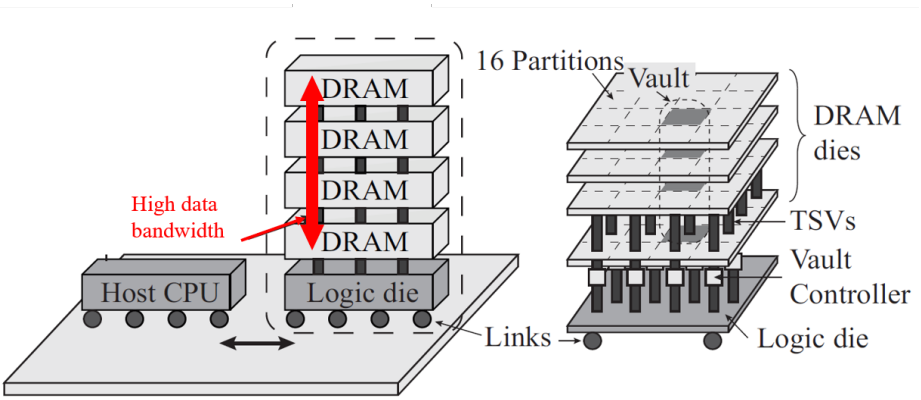


Figure 2.3: The architecture of Neurocube. Image is modified from the original work [17].

baseline [12]. Alternatively, FPGA has been explored as DNN accelerator for prototype verification purpose due to its flexibility. [15].

2.3.2 Near-memory processing accelerator

To further mitigate the memory wall issue and enhance the data transmission efficiency, several recent works investigate the near-memory processing architecture in which the logic die (tiled of processing engines) with the high speed memory in a 3-D stack [17, 18], as shown in Figure 2.3. Neurocube proposes a scalable architecture based on 3-D high bandwidth memory integrated with logic tier at the bottom for efficient DNN acceleration [17]. Deeptrain further extend the idea of Neurocube with enhanced dataflow and training support [18].

2.4 Processing-in-memory architecture

While memory-rich ASIC design and near-memory processing help to reduce the off-chip memory access latency and cost, these architecture still have separate memory and logic. To further reduce the cost of data movement, processing-in-memory (PIM) based design is extensively researched recent years [25, 26, 27, 28, 29, 30, 31]. Distinguished from the conventional hardware accelerator, the PIM architectures are designed to perform vector-matrix-multiplication (VMM) within the memory (i.e. VMM-in-memory). This section

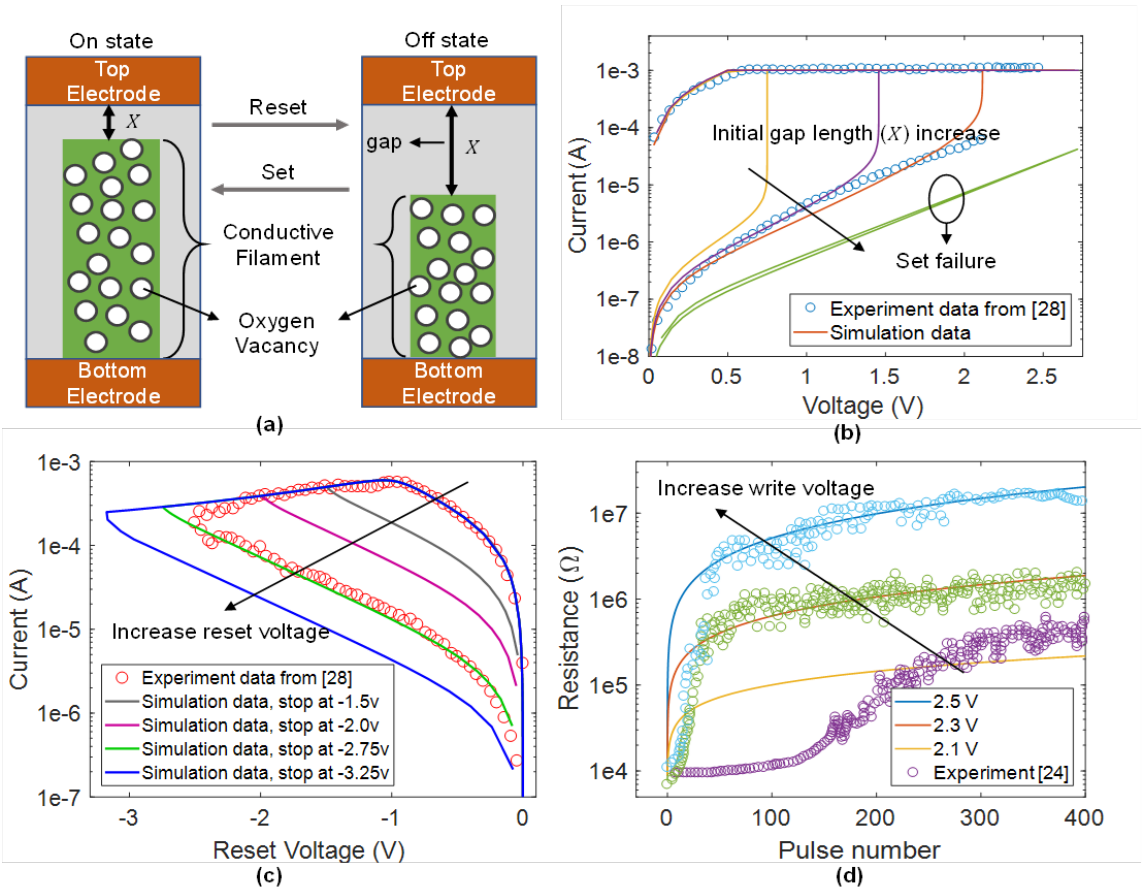


Figure 2.4: (a) Simplified ITR resistive switching model. (b) Device SET operations with different initial gap length. (c) Device RESET operations with applied voltage. (d) Resistance modulation with RESET pulses. The pulse width is 50ns. Experiment data are from [51, 52].

first introduces the most popular non-volatile memory candidate for PIM architecture: ReRAM. Then the basic concept for PIM architecture is discussed.

2.4.1 ReRAM

A ReRAM device consists of three layers. A resistive switching layer (e.g. HfO_x, NiO, TiO_2, Al_2O_3 or their combination) sandwiched between top and bottom electrodes (e.g. Pt, TiN) [51]. Device resistance can be modulated via applying set or reset voltage. A lot of works have been done to explore physical resistive switching mechanism of ReRAM [53, 54, 55]. In this work, we employ the Ion transportation recombination (ITR) model

Table 2.1: Resistive switching model.

Description	Equation
CF growth speed	$\frac{dg(t)}{dt} = v_0 \cdot \exp\left(\frac{-E_a}{k_B T(t)}\right) \cdot \sinh\left(\gamma \cdot \frac{q\varepsilon(t)a_0}{k_B T(t)}\right)$ (a)
Electric field at gap region	$\varepsilon(t) = \frac{V(t)}{g(t)}$ (b)
Temperature enhancement	$T(t) = T_0 + R_{th} \cdot I(t) \cdot V(t)$ (c)
Current conduction	$I(t) = I_0 \cdot exp\left(-\frac{g(t)}{G_0}\right) \cdot sinh\left(\frac{V(t)}{V_0}\right)$ (d)

Parameter description: g is the gap length, E_a is the activation energy (migration barrier), k_B is the Boltzmann constant, v_0 is the vibration frequency of oxygen atom. Here, we use it as a fitting parameter. a_0 is the atomic spacing. q is the unity charge. V is the voltage applied on electrode and I is the current flow through the ReRAM device. R_{th} is the thermal resistance of the resistive material. T_0 is the room temperature, and 300K is used in simulation. ε is the electrical field intensity. We assumed that the voltage on the gap area is dominating and the voltage applied on the conductive filament is neglected. We further assumed that the voltage applied on the gap area is uniform and thus the ε can be calculated via equation (b). I_0, G_0, V_0, and γ are fitting parameters.

[53]. In ITR model, the complex ion transportation and recombination process is simplified into the growth and rupture of Conductive filament (CF) in the resistive layer. Resistance of device is controlled by the size of the gap (X in Figure 2.4 (a)) between the electrode and the tip of the CF.

A simplified physical mechanism model, based on the ITR theory for bipolar ReRAM, is adapted in this research from prior works [53, 52, 56]. The details are shown in Table 2.1. The model is calibrated with existing experimental data [51, 52], shown in Figure 2.4 (b-d). As this model captures the dynamic behavior of the ReRAM switching, it helps to analyze the write latency and energy. We should note that with different parameters value (such as different initial gap length), the same model can be used to fit experiment data from different devices. As illustrated in Figure 2.4 (b-d), device in [52] ($TiN/TiO_x/HfO_x/TiO_x/HfO_x/Pt$ structure) shows very different on/off resistance compared with [51] ($TiN/HfO_x/Pt$ structure). With simple parameter tuning, this model can fit the experiment data from both works very well.

Table 2.2: Comparison between memory techniques.

Memory	DRAM	SRAM	ReRAM	FeFET
Data storage	Capacitor	Latch	Resistor	Transistor
Location	Off-chip	On-chip	On-chip	On-chip
Nonvolatility	Volatile	Volatile	Non-volatile	Non-volatile
Refresh	Yes	No	No	No
Cell type	1T1C	6-T/8-T	1T1R	1T
Cell area (28nm)	0.030 um^2	0.12 um^2	0.031 um^2	0.030 um^2
Access latency	> 100 ns	2.5 ns	< 5 ns	< 10 ns
Write energy*	~ 20 fJ	~ 100 fJ	~ 5 pJ	~ 1 fJ
Write latency*	~ 1 ns	0.25 ns	10 – 100 ns	50 – 500 ns

*Write energy only consider the energy consumed in memory cell;
* Write latency is evaluated in terms of writing an 256x256 array;

2.4.2 Other memory solutions for PIM architecture

Various memory techniques have been explored for PIM based designs including DRAM [28], eDRAM [10, 11], SRAM [29], ReRAM [25, 26, 27], and FeFET [30] as summarized in Table 2.2. *DRAM* based design mainly benefits from it's high density and large memory capacity. But as DRAM is difficult to integrate with digital CMOS, a full PIM architecture for DNN acceleration cannot be realized. The *eDRAM* support CMOS integration, but lacks density. Moreover, DRAM/eDRAM need periodic refresh introducing latency and design overhead, The *SRAM* provides faster speed and lower read/write power, but suffers from less density and large leakage power. Moreover, DRAM, eDRAM need periodic refresh and write back after every read (disruptive read issue), introducing latency and design overhead, particularly for DNN operations when weights are stored for a long time during inference.

More recently, *FeFET*, a transistor device with a ferroelectric oxide layer embedded in the gate dielectric [57, 58, 59, 60], have shown promise as a binary bit-cell for VMM designs [30]. This research mainly explores FeFET as a replacement for ReRAM to achieve better performance. Please refer to Chapter 4 and 5 for more details of FeFET.

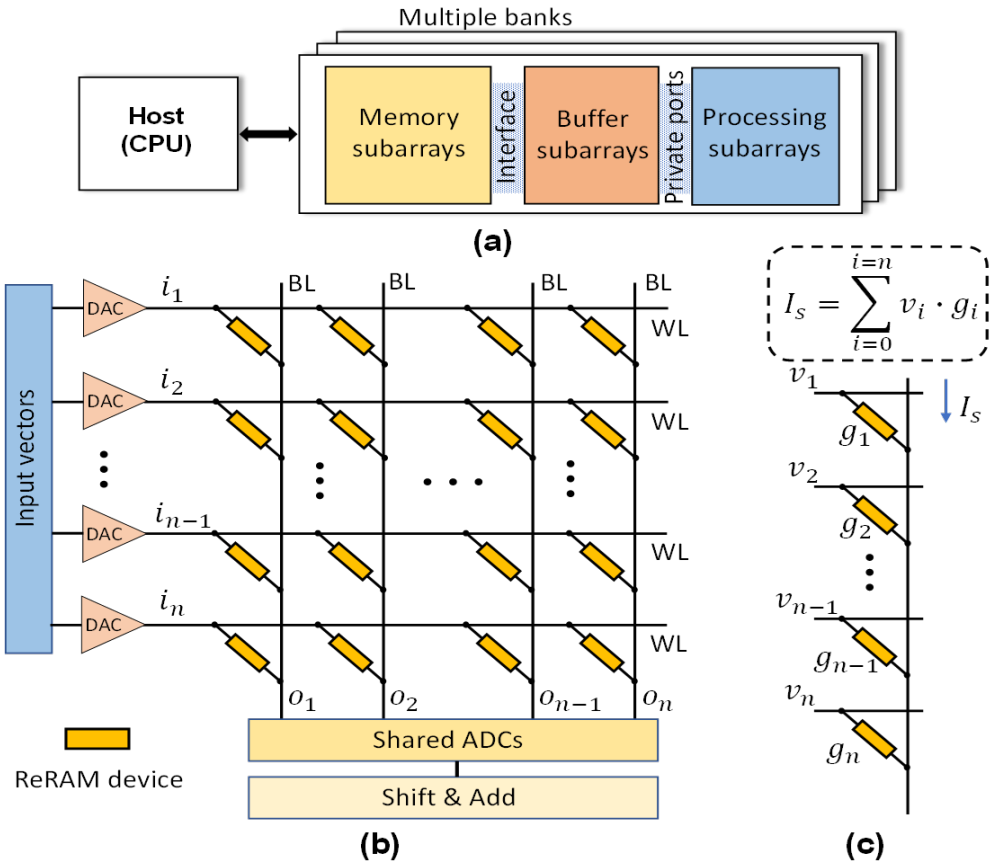


Figure 2.5: (a) PIM architecture. (b) ReRAM crossbar for matrix-vector multiplication. (c) Current is summed at bitline.

2.4.3 PIM configurations

It has been demonstrated that some emerging technology based non-volatile memory such as ReRAM, Phase change memory (PCM), and spin-transfer torque magnetic RAM (STT-MRAM) can perform logic operation beyond storage [25, 27, 26, 61, 62]. This unique feature allows them to serve for both computation and memory, enabling the PIM architecture. A commonly used PIM architecture is shown in Figure 2.5 (a), a memory bank is divided into three segments: memory sub-arrays which serves as conventional memory; buffer sub-arrays which serves as data buffer; and processing sub-arrays which are utilized to process the data. Private data ports are introduced between buffer sub-arrays and processing sub-arrays for high data bandwidth. Further, this private data bus is isolated from main memory

bus, hence, it does not consume the bandwidth of main memory. As an energy-efficient technique, PIM best exploits the data locality by re-configuring part of the memory blocks as processing units. Moreover, data can be directly accessed and transmitted internally without any external memory interfaces. PIM provides large memory bandwidth and fast data access, also reduces the data moving energy, which are very critical to data demanding applications such as machine learning. Among all the candidates, ReRAM based PIM architecture attracts lots of research attentions since ReRAM has fast writing speed, CMOS compatibility, high on-off ratio and analog feature, i.e. multi-level cell (MLC) [51, 32]. Recent works have demonstrated that ReRAM based PIM architectures achieve orders of magnitude speed and energy efficiency improvements compared with the GPUs and ASIC designs on CNN acceleration [25, 27, 26].

The core component of ReRAM based accelerator is a crossbar based processing engine for matrix-vector multiplication which is the main computation for machine learning, Figure 2.5 (b). During computation, neural network's parameters are first programmed into the ReRAM devices. Then input vectors are fed into the crossbar array as wordline (WL) supply voltage. As shown in Figure 2.5 (c), based on Kirchhoff's law, the current summed at each bitline (BL) results the matrix-vector multiplication. Since the computing inside the crossbar array is in analog fashion, DAC and ADC are required for the WL/BL peripherals. Further, due to limited storage capacity of single ReRAM device, multiple cells connecting to the same WL are used to represent one parameter value. For example, 8 ReRAM devices are utilized to represent one 16-bit number with each cell stores 2 bits. Shift & add blocks are used to sum up the computation result from different columns. Further, it is not practical to implement a 16-bit DAC for each row of ReRAM crossbar. Therefore, the input number is divided into several segments and sequentially fed into the network. For instance, 16 clock cycles are needed for a 16-bit input value with 1-bit (the read voltage is either 0 or 1) per cycle. Note that one-resistor-one-transistor (1T1R) structure is now widely used for ReRAM crossbar design to increase the selectivity and reduce the leakage

current [35, 63]. Figure 2.5 only shows the ReRAM cell for simplicity.

CHAPTER 3
RERAM BASED PIM ARCHITECTURE FOR RECURRENT NEURAL NETWORK

3.1 Challenges in ReRAM based PIM design for RNN acceleration

Even though several architecture works have explored ReRAM based CNN acceleration [25, 26, 27], none of these works has considered utilizing ReRAM to accelerate the computing of RNN. Similar with CNN, the most expensive operation of RNN is the matrix-vector multiplication, however, RNN configurations such as LSTM [39] and GRU [40] also contain the element-wise multiplication which is not feasible to be implemented with ReRAM crossbar. Further, due to the time dependence, in RNN's computation, the processing for current input must wait the hidden state calculated from previous step, therefore, the pipeline architecture for ReRAM based CNN accelerator is not suitable for RNN computing.

In the early stage of this research, we proposed a ReRAM based accelerator design for basic RNN [64]. However, that design is insufficient in several aspects: first, it only supports the basic RNN but not feasible to compute LSTM or GRU, which constrains the wider utilization; second, the computation for non-linear function is performed by the analog neuron circuit which can introduce non-negligible errors; third, the design is not scalable since it only focuses on crossbar level modeling while the architecture and system design are missing.

3.2 System design

In this section, the system architecture overview is first presented. Then the function of each sub-block in the processing engine is discussed. At last, the dataflow and pipeline are

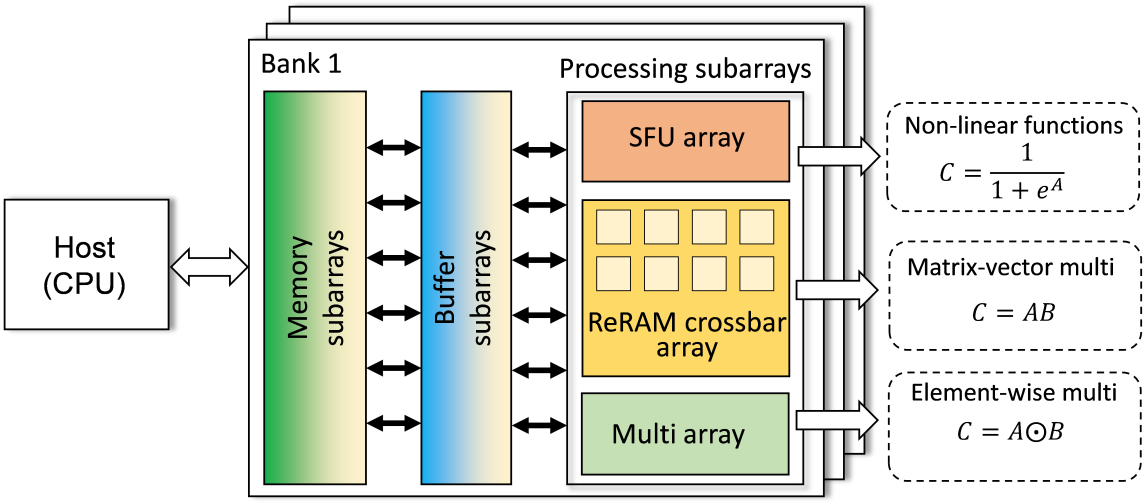


Figure 3.1: System architecture overview.

presented. The implementation details for the sub-blocks are presented in section 3.3.

3.2.1 System architecture overview

Figure 3.1 presents an overview of the proposed system architecture. Similar with prior PIM architectures, the memory bank is divided into three partitions, including the memory sub-arrays, buffer sub-arrays, and processing sub-arrays. One thing to note is that even though the processing arrays can also be used for data storage like in PRIME [25], we observe that the redesigned array peripherals impose extra data access latency, therefore, the processing sub-arrays are dedicated for computing in our design. As mentioned earlier, existing ReRAM PIM architectures are not suitable for accelerating RNN computation. To make it RNN friendly, we further divide the processing sub-arrays into three segments: ReRAM crossbar array for the matrix-vector multiplication; special function unit (SFU) array for the non-linear functions; and multiplier array handling the element-wise multiplication.

3.2.2 Details of system design

Figure 3.2 shows a detailed system architecture and the WL/BL peripherals design.

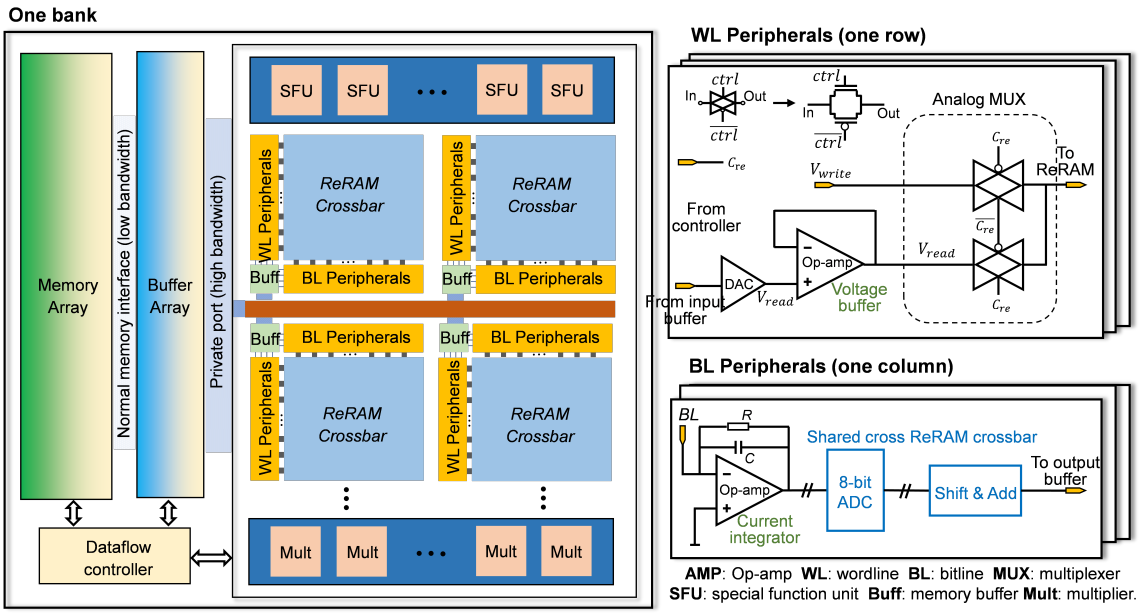


Figure 3.2: System architecture and the WL/BL peripherals design.

The wordline peripherals support both reading and programming. The switching between reading mode and programming mode is realized by an analog multiplexer (MUX) designed with two transmission gates, as shown in Figure 3.2. To be more specific, the behavior of the analog MUX is controlled by the read enable signal (C_{re}) from dataflow controller. When C_{re} is enabled, the MUX is transparent to read voltage from the read voltage buffer (design with an Op-amp configured in voltage follower mode) and blocks the write voltage. The read voltage is generated from the DAC which transfers the input digital data into analogy voltage. On the other hand, when the C_{re} is low, the MUX is transparent to write input signal and blocks the read voltage input. The write signal contains a series of identical programming pulses which will be further discussed in later section.

Bitline peripherals. The first stage of BL peripherals computation is to sample the current from each column, convert it into voltage and hold it for later analog to digital conversion. This can be accomplished by an Op-amp based sample and hold circuit. Note that the same Op-am in wordline peripherals can be reused here with a capacitor inserted in the feedback loop (Op-amp configured in current integrator mode rather than voltage

follower mode [65]). In next stage, ADC is required to convert the analog voltage signal to digital value. Note that one ADC is shared for all the columns in a crossbar because ADC consumes a lot of power and area [26, 66]. A shift add unit (SAU) is integrated after the ADC to accumulate data from different BLs and different cycles. Finally, the digital output will be temporarily stored at the local buffer and then collected back to the buffer sub-array.

Special function unit (SFU) and Multiplier. Even though ReRAM crossbar array can efficiently perform computation for matrix-vector multiplication, it is not feasible to solve complex mathematical equations such as sigmoid function (σ), hyperbolic tangent function ($tanh$), and rectified linear unit (ReLU) which are widely used in neural networks serving as activation function. Thus, we implement SFU to handle the computation for complex mathematical functions. Element-wise multiplication is another important operation of LSTM and GRU. Therefore, unlike the ReRAM based CNN accelerators, our design implements an additional multiplier array to handle the element-wise multiplication operation.

Local buffers. The purpose of local buffers is to serve as the L1 cache for the ReRAM crossbar based computation engine. During inference, they receive data from buffer sub-arrays and send the data to WL peripherals. They also collect computing results from BL peripherals and send them back to buffer sub-arrays. Local buffers are distributed and placed close to crossbar arrays. The addresses for read/write are controlled by dataflow controller and are identical for all local buffers.

Dataflow controller. The dataflow of the proposed system is controlled by the dataflow controller. On the top level, it coordinates the dataflow between memory sub-array and buffer sub-array, it also manages the data transmission between the buffer sub-array and local buffers. During reading (i.e. inference), the dataflow controller enables the read enable signal C_{re} for all crossbars. Therefore, the reading voltages can drive the corresponding WLs while the programming voltage is blocked. On the other hand, during writing, the C_{re} signal to one specific ReRAM crossbar array is disabled to let the writing voltage get

through while all the rest arrays are set in idle mode. Therefore, the writing operation is crossbar-wised while the reading can be performed in multiple crossbars simultaneously.

3.2.3 Dataflow and pipeline

One of the unique features of RNN computation is the data dependency, i.e. the computing in current step relies on the results of previous step. Therefore, the dataflow and pipeline of ReRAM based CNN accelerator is not suitable to accelerate RNN computing. In this section, we first illustrate how to map RNN computing to our design and then propose a RNN friendly pipeline execution framework. We use LSTM as an example. Analyses for the basic RNN and GRU should follow the same procedure.

Before mapping matrices-vector multiplication to ReRAM crossbar array, we need to re-organize the input and parameter matrices. Take equation 3.1 as an example, two pairs of matrix-vector multiplication are concatenated into one:

$$x_t U^i + h_{t-1} W^i + b_i == \begin{bmatrix} x_t & h_{t-1} \end{bmatrix} \begin{bmatrix} U^i \\ W^i \end{bmatrix} + b_i \tag{3.1}$$

Note that adding the bias b_i to the matrix-vector multiplication is trivial since we can use a row of ReRAM cells to store the bias value and force the read voltage to be 1. Figure 3.3 (a) shows the matrix-vector operation after concatenation. Here we assume the input vector size is 2 and hidden state vector size is 3. After concatenation, parameter matrix (concatenation of U and W) are programmed into ReRAM cells and input matrix (concatenation of x_t and h_{t-1}) are used as WL input, shown in Figure 3.3 (b). For simplicity, in Figure 3.3 we assume that each cell can store one parameter value. In practice, due to limited device precision, multiple adjacent cells in the same row are used to store one element and the input signals are also partitioned into several segments and fed into the network sequentially. After computation, results will be first temporary stored in the local buffer and then transmitted back to the buffer sub-arrays.

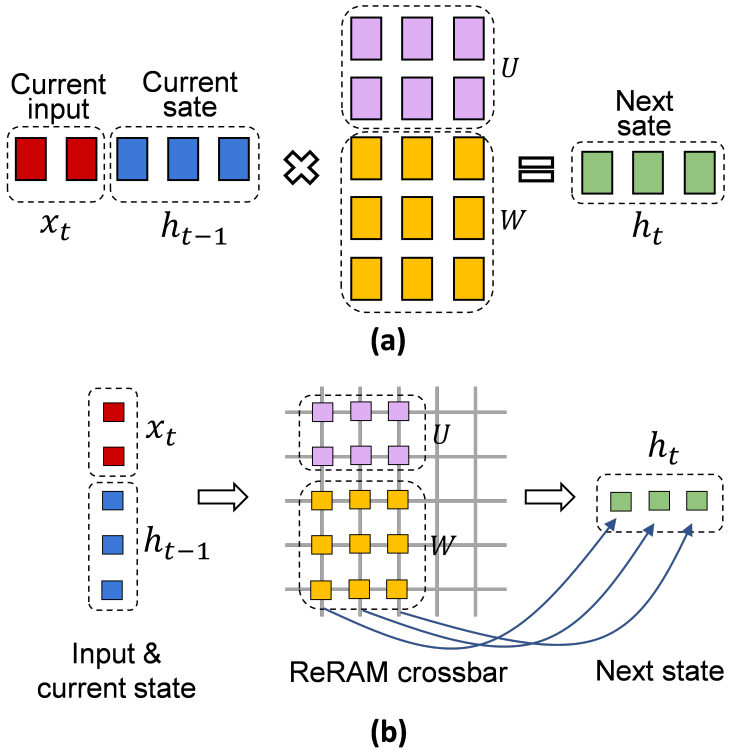


Figure 3.3: (a) Matrix-vector multiplication after concatenation. (b) Mapping the computing to ReRAM crossbar array.

The computation for element-wise multiplication and non-linear functions are relatively straightforward. The dataflow controller fetches the data from buffer sub-arrays to SFU/multiplier array and then collects the results back to buffer arrays.

An RNN friendly pipeline execution flow is developed to increase the system throughput, shown in Figure 3.4. Assume we have three input sequences (A, B, and C) which will be processed by the LSTM network. For example, A_1 to A_6 are the first 6 words in a sentence and we want to predict the 7th word based on them. At the first clock cycle T_1, input vector A_1 and the hidden state vector h_0^A calculated from previous stage are fed into the ReRAM crossbar for matrix-vector operations. In next cycle T_2, input vector from another sequence B and the corresponding hidden state h_0^B are fed into the crossbar array; meanwhile, the result of matrix-vector multiplication of A_1 and h_0^A are sent to SFU arrays to perform non-linear function operations. Then, in the third cycle T_3, temporary results for sequences A and B are moved forward to the next step, and sequence C starts the matrix-

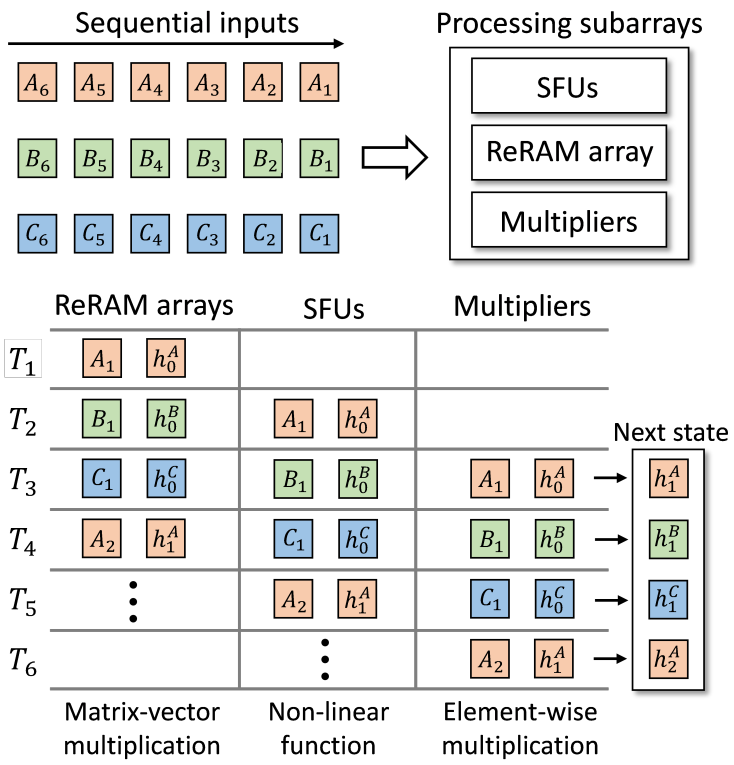


Figure 3.4: Three stages pipeline for RNN computation.

vector operation. At the end of T_3, the new hidden state h_1^A for sequence A is generated. At the next clock cycle T_4, the second input vector A_2 from sequence A can be fetched to the processing engine since the hidden state h_1^A is ready. With the proposed pipeline, all the processing units in the system are busy, therefore, the maximum throughput can be achieved.

3.3 Detailed system implementation

This section first presents how to enable re-programmability (i.e. device programming), then the circuit level implementation of the sub-blocks is discussed. To get an accurate estimation for power and area, the digital design (including dataflow controller, multiplier, and SFU) is synthesized with Design Compiler and PrimeTime based on 28nm CMOS technology from Synopsys [67]. We also run SPICE simulation for the analog sub-blocks (including Op-amp and analog multiplier) with 32nm PTM model (28nm model is not

available in the PTM library) [68].

3.3.1 Programmability

Efficient programming is always challenging for ReRAM crossbar array as the write operation normally requires much longer time and consumes more energy than reading [69]. Further, during reading and inference, all the ReRAM devices are involved. Devices connecting to the same WL receive the same input voltage. Therefore, reading can be performed with 1 read clock cycle. However, during writing, each cell needs to be programmed differently and might have diverse values compared with its neighborhood. Thus, multiple clock cycles are required to program an array. Array-level parallel programming scheme has been proposed in prior work [70]. However, the complexity of peripheral circuit design and the ultra-high power density make it prohibitive for large-scale implementation.

In our design, the programming is performed in a column-wise fashion with a series of identical pulses, i.e. the ReRAM cells connecting to the same BL are programmed simultaneously. Fig. 8 shows the programming scheme for a two arrays system. To program the top ReRAM array, the array selecting signal C_as is enabled for this array while disabled for the bottom one. Since we only have two arrays, the C_as has two bits, one for the top array, the other for the bottom array. The column selecting signal C_cs is used to determine which column to be programmed by turning on the selector transistors of the corresponding column via the select-line (SL). Similarly, C_cs also has 2 bits. We should note that this can be easily extended to a multiple arrays system and larger array size simply with more bits of C_as and C_cs. As shown in Figure 3.5 (a), controlled with C_cs and C_as, only the cells in the first column of the top array can be programmed while all the other cells are disconnected from the network.

Moreover, ReRAM cells can be programmed into different states, controlled by the number of pulses it receives, as illustrated in Figure 3.5 (b). The shared pulse generator consistently output identical pulses (same amplitude and length). The pulse counter, initial-

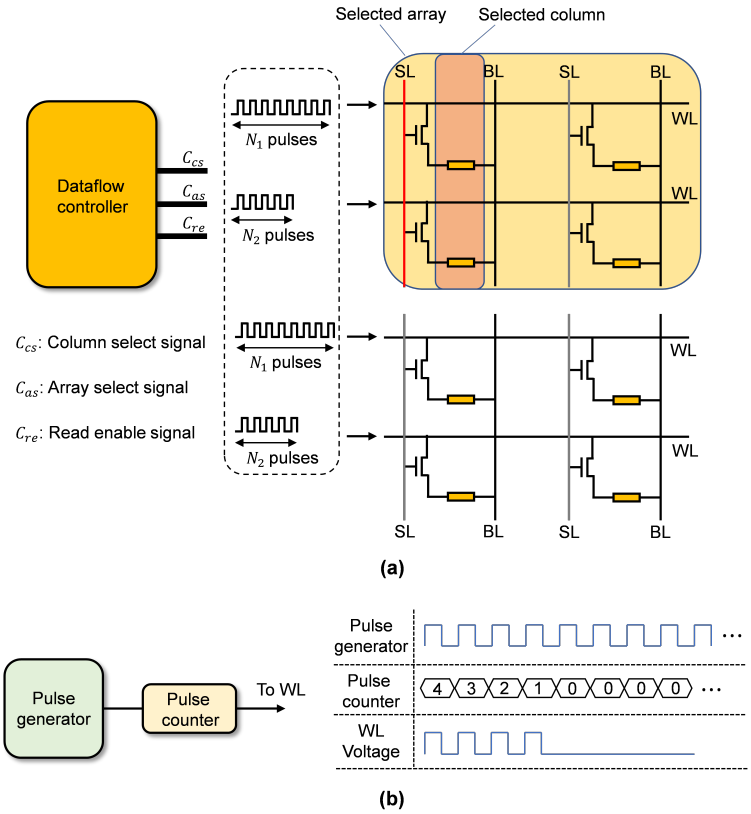


Figure 3.5: The programming schemes. (a) One column of a crossbar is programmed simultaneously. (b) Pulse series based programming. The conductance change is determined by how many pulses the WL received.

ized with the total number of pulses, counts down once when receiving one pulse. When it counts down to zero, it will block the pulse to be sent to the WL. We should note that the bottom array is disconnected from the network by the array select signal, the cells won't receive the writing pulses.

With the proposed writing scheme, it takes N writing cycles to program a $N \times N$ ReRAM crossbar array. Note that typically the writing is much slower than reading, therefore, it is highly desired that no re-programming occurs during computing. In section V, we show that the performance will significantly drop when the neural network size is larger than the system capacity (i.e. re-programming is required).

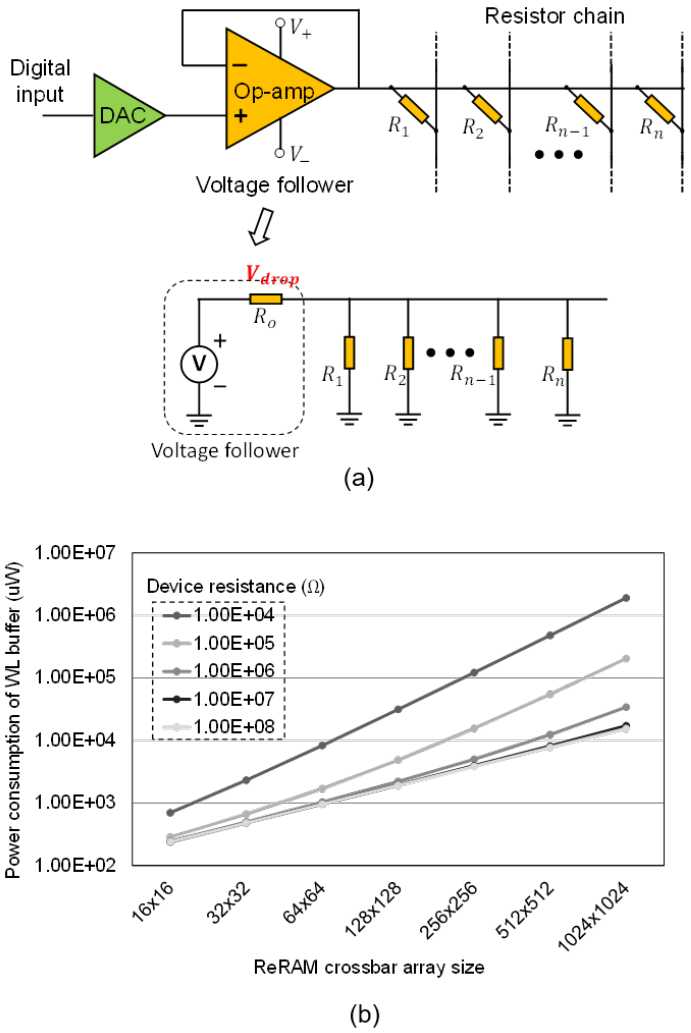
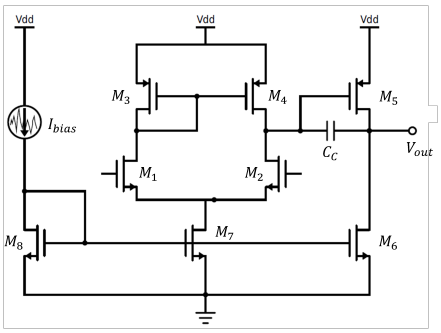


Figure 3.6: Wordline output voltage drop due to limited driving ability: (a) circuit schematic, and (b) Op-map power scaling for different ReRAM crossbar size.

3.3.2 Woldline driving ability

A major circuit design challenge of ReRAM accelerator is the WL driving ability issue. It is unrealistic to directly use the output of DAC or other digital circuits to drive the wordline of ReRAM crossbar since they can't provide enough current to drive resistive load. A common approach to increase the driving ability is to insert a buffer stage between the DAC and crossbar array [65]. The buffer can be designed with an operational amplifier (Op-amp) configured at voltage follower mode, as shown in Figure 3.6 (a). The driving ability is determined by the Op-amp output stage bias current (i.e. transistor size). Increasing the



Op-map parameters

Technology	32nm PTM
Slew Rate	20V/us
Supply voltage	1.8
ICMR	0.8v~1.6v
DC gain	84dB
GBW	30MHz
Phase margin	40°
Power	15uW

Figure 3.7: Two stages Op-amp design. Key parameters are listed in the table.

output stage transistors size can reduce the output impedance therefore alleviate the read voltage drop effect. Figure 3.6 (b) shows the Op-amps power to drive different ReRAM crossbar array size with less than 10% read voltage drop considering various device resistance. We observe that the power consumption of Op-amp increases proportionally with array size and device conductance (i.e. the reverse of device resistance). Therefore, to reduce the power and area overhead of WL buffer, we constrain the ReRAM array size to be 128×128. And, in the rest of this paper, we assume the on-state resistance is 1 MΩ (i.e. LRS is 1 MΩ). We should note that device with high resistance has been reported in prior works with a tradeoff of other device characterizations such as low on/off ratio and worse uniformity [32, 70].

3.3.3 System implementation

Most of the prior ReRAM based accelerators focus on the architecture design but lack detailed circuit analysis. We carefully design the peripherals and control logic to get much accurate modeling for the system area, energy efficiency, and throughput.

Op-amp design: Op-amp is the most important component because it is the key component of the WL driver (Op-amp configured in voltage follower mode) as well as the sample and hold circuit (Op-amp configured in current integrator mode). To reduce the power consumption and area, we design a simple two-stage Op-amp, as shown in Figure 3.7. Note that the data in Figure 3.7 are based on the SPICE simulation results of the min-

imum size Op-amp design which is employed for the BL current-to-voltage converter. To ensure the WL voltage buffer has enough driving ability, the Op-amp output stage transistor size must be scaled up according to the crossbar size and ReRAM resistance.

DAC: The design of DAC is trivial since 1-bit DAC is just a simple voltage buffer. In this paper, we divide the 16-bit fixed point number into 16 1-bit numbers and sequentially feed them to the ReRAM crossbar arrays. This is similar with prior works and has been proved as a sweet spot for ReRAM based accelerator [26, 27]. Therefore, to compute the matrix-vector multiplication for 16-bit numbers, 16 cycles are required. For multi-bit DAC, resistor ladder can be used [71].

ADC: Compared with DAC, the design of ADC is non-trivial and has been reported as the main bottleneck for computing speed and introduces great power and chip area overhead [26, 72]. We employ an 8-bit 1.2 GS/s (Giga samples per second) successive approximation register (SAR) ADC design which is optimized for energy efficiency and area (the power consumption is 3.1 mW with chip area of $0.0015\ mm^2$ using 32 nm CMOS technology) [66]. As mentioned earlier, 1 ADC is shared for an entire ReRAM crossbar array to reduce the energy/area overhead. Our simulation indicates that ADC is also the major bottleneck of the system performance in our deisgn. Further, it introduces significant power (43% of system power) and area (49% of the chip area) overhead.

Digital sub-blocks: There are several digital circuits in our design, including the multiplier, SFU, shift add unit and the dataflow controller. The multiplier in our design supports 16-bit fixed point number operation since it has been demonstrated that 16-bit is sufficient for most machine learning applications [12].

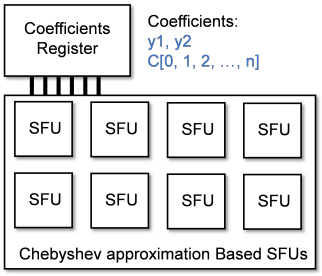
There are multiple different approaches for the SFU design including Taylor expansion based approximation, Chebyshev approximation [73], and piece-wise linear function (PWL). As illustrated in Figure 3.8 (a), we compare the accuracy and synthesized power/area number of these approaches. The accuracy is measured using sigmoid function (a popular non-linear function used in neural network). And the synthesis results are from

	Taylor expansion	Chebyshev	PWL-1	PWL-2
Accuracy*	2.5e-3	6.4e-4	1.1e-2	4.8e-4
# of intervals	10	10	10	50
# of coefficients	30	50	20	100
Power (uW)	1273	1688	1068	2904
Area (um^2)	4009	5212	3028	9357
Clock frequency	500 MHz			

(a)

```
d1=0.0; // temporary variable
d2=0.0; // temporary variable
// approximation range [a, b]
y1=(2.0*x-a-b)/(b-a);  // y1 is parameter
y2=2.0*y1; // y2 is parameter
// n is the number of coefficients
for (j=n-1; j>0; j--)
   sv=d1; // temporary result
   // c[0, 1, 2…] are coefficients
   d1=y2*d1-d2+c[j+1];
   d2=sv;
end
result=y1*d1-d2+0.5*c[1]; // final result
```

(b)



(c)

Figure 3.8: (a)Comparison of different SFU designs. (b) Pseudo code for Chebyshev approximation. (c) SFU array and coefficients register.

28 nm CMOS technology. With same number of intervals (10 interpolations in the input range [-5, 5]), the Chebyshev exhibits the best accuracy but consumes more power and area since it has more coefficients and computation. PWL approach (PWL-1) have lowest power and smallest area but the accuracy is not good enough. With more coefficients (i.e. interpolations) (PWL-2), we observe comparable accuracy with Chebyshev approach, but the power/area are significantly higher since it has much more coefficients.

In this work, the Chebyshev approximation is employed since it provides the best accuracy with moderate power and chip area overhead. The procedure of Chebyshev approximation is shown in Figure 3.8 (b). The coefficients are first calculated with CPU and pre-loaded into the local register file. During computing, SFU will access the register file and calculate the non-linear function, as shown in Figure 3.8 (c).

In prior ReRAM accelerator works, circuits are designed to solve a specific function [25, 26]. Compared with them, the SFU based approach is advantageous since it has more flexibility, can be easily reconfigured (i.e. load different coefficients) to solve different functions.

Table 3.1: Power consumption and area for sub-blocks in the proposed design.

Component	Power (mW)	Area (um^2)	Number
WL peripherals	1.9	40.9	128
BL peripherals	5.0	1550	
Local buffer	0.16	648.0	
ReRAM array	0.01	147.5	
Dataflow controller	0.3	300	1
SFU	1.68	5212	16
Multiplier	0.18	1155	4
Bank total	932.86	0.39 mm^2	N/A

Table 3.2: Latency of sub-blocks in the proposed design.

Components	Latency (ns)	Throughput
Matrix-vector Mult	100	163.8 GOP/s
Non-linear Func	5	3.2 GOP/s
Element-wise Mult	1	4.0 GOP/s

We should note that, ReLU based non-linear function is the most commonly used one in recent DNN development which can be easily solved with very simple logic (no need for SFU). However, in this work, SFU is still included to ensure the flexibility.

Memory array and local buffer: A dedicated in-house design simulator is used to model the read/write power and latency based on the experimental data calibrated ITR model [53, 52, 56]. The size of 1T1R structure is assumed as $10F^2$ where F is the minimum lithography length in 28nm technology. Similar with prior works [10, 26], eDRAM are employed as local buffer. The power and area of eDRAM is modeled based on [26].

The power consumption and area for each sub-block are listed in Table 3.1. We also analyze the latency of each step of computation, shown in Table 3.2. Note that the throughput for matrix-vector multiplication (ReRAM crossbar arrays) are much higher than non-linear functions and element-wise operations since in RNN computing, the matrix-vector operation is still dominating.

Table 3.3: Benchmarks.

Datasets	Description	Networks	Hidden state	*Ops/frame
NLP	Predict next word from previous input words	*bRNN	128	$5.6x10^5$
		LSTM		$2.2x10^6$
		GRU		$1.7x10^6$
HAR	Classify human activity from 6 categories.	*bRNN	128	$2.2x10^6$
		LSTM		$9.0x10^6$
		GRU		$6.7x10^6$
MNIST	Hand written digits classification	MLP	256	$3.9x10^5$
		CNN	/	$9.8x10^6$

***bRNN:** basic RNN **Ops/frame**: number of operations per input sample

3.4 Results analyses

3.4.1 Experiments setup

Benchmark: We evaluate the system performance with two RNN based applications. One is NLP where we want to predict the next word in the sentence based on previous input vocabularies. The dataset is available at [74]. The Second application is human activity recognition (HAR) and we use the dataset from [75]. For both applications, we evaluate the performance with three RNN configurations: basic RNN, LSTM, and GRU. For all networks, the number of hidden layer features are 128 (i.e. the length for hidden state vector is 128). Should aware that our system can also be utilized to accelerate the computing for feedforward neural networks such as Multi-layer perceptron (MLP) and CNN. Current design doesn't support max pooling. However, we argue that the computation for pooling layer is trivial since it is just a simple comparison logic. Hence, we also include MLP/CNN into our benchmark. The benchmark MLP has one hidden layer with 256 hidden neurons. The CNN contains 2 Convolutional layers, and 1 fully connected layer. The benchmark information is summarized in Table 3.3.

GPU baseline: We perform the experiments with Tensorflow deep learning framework [76] running on a state-of-the-art NVIDIA GPU. The GPU parameters are listed in Table 3.4.

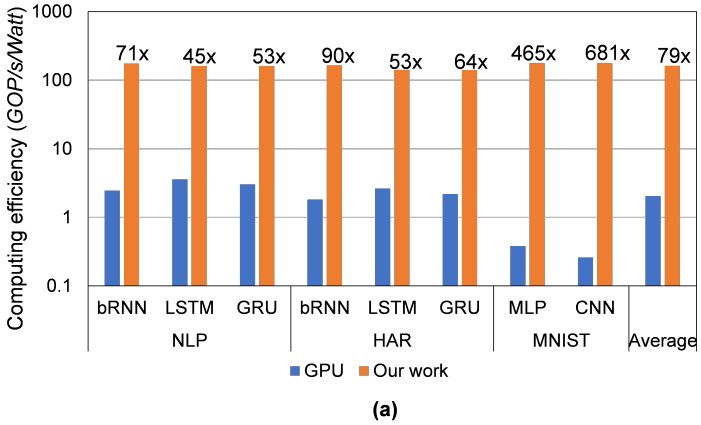
Table 3.4: GPU parameters.

Name	NVIDIA GeForce GTX 1080Ti
Global memory	11 GB GDDR5X
CUDA cores	3584
GPU clock	1.68 GHz
Memory clock	5.4 GHz
Memory bandwidth	320 GB/s
Cache size	112 KB L1 cache/2.75 MB L2 cache

On-chip resource utilization Using the NLP based LSTM network as an example, we get 4 parameter matrices [U^i,W^i], [U^f,W^f], [U^o,W^o], and [U^g,W^g] based on the equations of LSTM. Given the input size (28) and hidden state size (128), the matrices sizes is 156×128. Therefore, the parameter size is 156 KB considering all parameters are represented with 16-bit fixed point number. Similar analysis can be applied to calculate the parameter size for other benchmarks. On the other hand, our system contains 128 arrays where each array contains 128×128 devices. The total capacity of our system is 512 KB considering each ReRAM device stores 2-bit. Our system is large enough to hold all the parameters for the benchmarks. For neural networks with large parameter size, multiple banks can be tiled together to increase the system capacity [26]. The tradeoff of tiling multiple banks together is that the system power consumption will increase proportionally, which is not feasible for low power platforms.

3.4.2 System performance and comparison with other platforms

The computing efficiency is evaluated and compared with our GPU baseline. The computing efficiency is represented in term of GOP/s/W (Giga operations per second per Watt). The GPU power is measured by nvidia-smi provided by NVIDIA CUDA toolkit. Figure 3.9 (a) shows the computing efficiency comparison considering different datasets and networks. On average, the improvement across all the RNN benchmarks are 79x. Interestingly, experiment results indicate that the GPU computing efficiency for MLP/CNN is



Frame rate (Fps) for benchmarks

Frame rate		GPU	Our work
NLP	bRNN	877.2 K	289.6 K
	LSTM	319.5 K	68.0 K
	GRU	359.7 K	88.0 K
HAR	bRNN	161.0 K	69.4 K
	LSTM	58.2 K	14.6 K
	GRU	64.6 K	19.8 K
MNIST	MLP	196.2 K	65.4 K
	CNN	57.4 K	25.4 K

(b)

Figure 3.9: (a) Computing efficiency in terms of GOP/s/Watt. (b) System throughput in terms of frame rate (Fps).

lower than RNN. This is mainly caused by input data access latency. To be more specific, the input data for feedforward networks all comes from external memory. Differently, input data of RNN are the concatenation of input data and hidden state calculated from previous stage. Since the hidden state are temporary data which only exist in GPU's L1/L2 cache, it can be directly fetched without the latency for accessing external memory. In our PIM architecture, both the input data and temporary hidden state are stored in the buffer sub-arrays, therefore, the performance for computation of RNN and MLP/CNN is similar. The second reason impacting the GPU performance for CNN computing is that the data structure of convolutional operation should be re-organized for matrix-vector multiplication. This overhead no longer exists in our system since the parameters are pre-loaded into the ReRAM crossbar arrays.

Should aware that the throughput of our system is less than the GPU, as shown in Fig 12(b). On the other hand, our system only consumes 0.6 Watt while the average GPU power is around 200 Watt, enabling more than 300x improvements for the energy consumption.

The computing efficiency of our system is further compared with prior works, including FPGA based LSTM accelerator, ESE [15]; ReRAM based CNN accelerators, ISAAC [26] and PipeLayer [27], and ASIC based CNN/RNN accelerator, DaDianNao [10] and EIE [16]. The results are summarized in Table VI. The FPGA based approach demonstrates the

Table 3.5: Comparison with other hardware accelerators.

Accelerator Name	ESE	ISAAC	PipeLayer	DaDianNao	EIE	Our work
Target networks	RNN	CNN	CNN	CNN	CNN RNN	CNN RNN
Technology	22 nm	32 nm	/	28 nm	45 nm	28 nm
Approach	FPGA	ReRAM	ReRAM	ASIC	ASIC	ReRAM
Training support	No	No	Yes	No	No	No
Parameter storage	DRAM	eDRAM	ReRAM (In situ)	eDRAM	SRAM	ReRAM (In situ)
Computing efficiency (GOP/s/W)	6.88	380.7	142.9	286.4	174.1	116.3

lowest performance because it stores the parameters in the external DRAM. Also, the maximum clock frequency for FPGA is much lower than the ASIC, which further constrains the performance. Compared with ReRAM based CNN accelerator, we demonstrate similar performance with PipeLayer but less than ISAAC. The reason is that prior works do not consider the driving ability issue and ignore the power consumption of the WL buffer which is one of the major energy hungry component (52.7% of the total power in our design). If ignore the power consumption on the WL buffer, our design can achieve performance with 341 GOP/s/W, which is similar with ISAAC. We show that the overheads associated with the peripheral circuit can significantly degrade the computing efficiency.

The ASIC approach achieves the state-of-the-art computing efficiency. The primary reason is that ASIC designs employ large size on chip memory to store the parameters; therefore, the data movement energy is reduced. On the other hand, the digital ASIC does not have the power overhead associated with the analog peripherals, namely, the WL drivers and ADC. The in-situ computation in ReRAM array enhances the efficiency multiply-and-accumulate (MAC) operation, however, the need for 16 cycles to feed the 16-bit fixed point data into the ReRAM array degrades the efficiency.

In ASIC designs, both eDRAM and SRAM are volatile memory, the stored data will vanish after power off, making it not suitable for platforms with limited energy budget such as mobile devices. On the contrary, ReRAM based design is advantageous since the

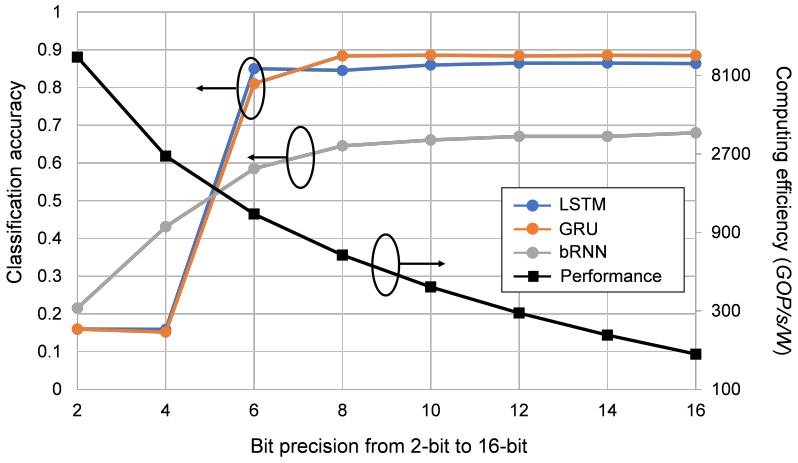


Figure 3.10: System performance with lower bit-precision.

parameters are stored in non-volatile memory. Therefore, we conclude that *if the computing is performed in datacenter where the power supply is sufficient and stable, ASIC based approach is preferred; if the computing is performed in a distributed low power platform, the ReRAM approach provides more benefits.*

3.4.3 Enhance performance with lower bit-precision

Previous analyses assume that all the parameters and input data are represented with 16-bit fixed point number. To further enhance the computing efficiency, we explore using lower bit precision for computing. Figure 3.10 shows the trade-off between computing efficiency and classification accuracy. The data in Figure 3.10 is based on HAR dateset [75]. Similar results can be observed for other dataset. Simulation indicates that 8-bit precision demonstrate satisfactory results while lower bit precision (less than 6-bit) show significant accuracy drop. This is consistent with the result in prior work [12]. With 8-bit precision, the performance (GOP/s/W) is 4 times higher than 16-bit.

3.4.4 Impact of device variation

The device variation of ReRAM can significantly deteriorate system performance. Device variation comes from the stochastic formation and rupture of conductive filament in

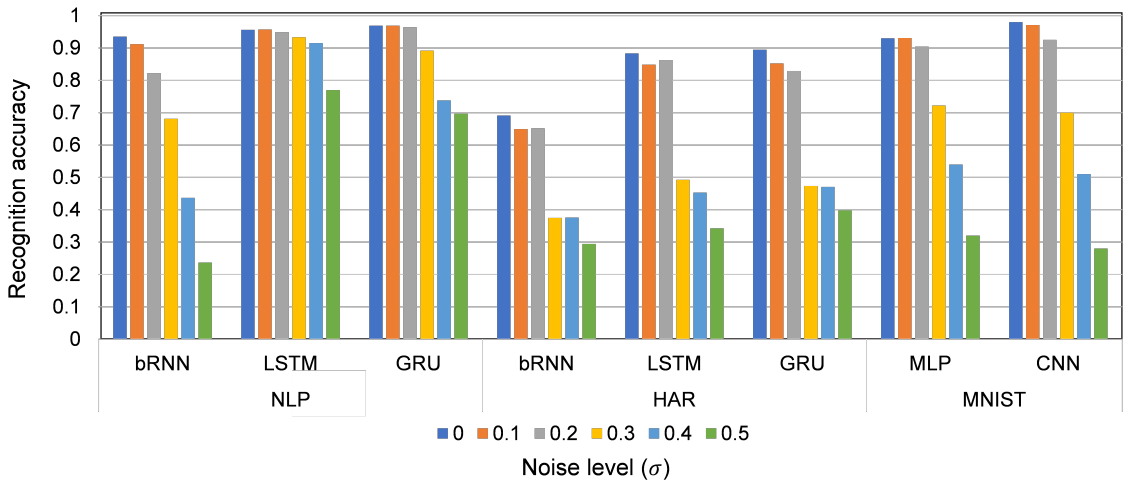


Figure 3.11: Computing accuracy with different levels of device variation.

the resistive layer of ReRAM (i.e., generation and recombination of oxygen vacancy is stochastic) [53, 56]. Variations exist in cycle-to-cycle operation and from one device to another device. Variations can be caused by read or write operation, properties of resistive materials, and various fabrication factors. In this work, we consider using Gaussian noise to represent the device variation. Other forms of device noises such as bit-flip error or random telegraph noise (RTN) can be analyzed in a similar way. We use the following equation to represent the Gaussian noise of device conductance:

$$g_{noise} = g_{ideal} \cdot (1 + N(0, \sigma^2)) \tag{3.2}$$

where g_ideal is the expected device conductance without variation; $N(0,^2)$ is the normal distribution with mean equals to 0 and standard variation σ. It has been measured that the variation is normally less than 0.2 [51]. We evaluate the computing accuracy in terms of the classification accuracy for the benchmarks. As shown in Figure 3.11, we observe that the accuracy drop is insignificant when the standard deviation of the added Gaussian is less than 0.2. However, the accuracy drops a lot for all the benchmark tests when the noise level is large. Data in Figure 3.11 also indicates that the performance of LSTM and GRU are better than the basic RNN especially for a more complex dataset. For example, with

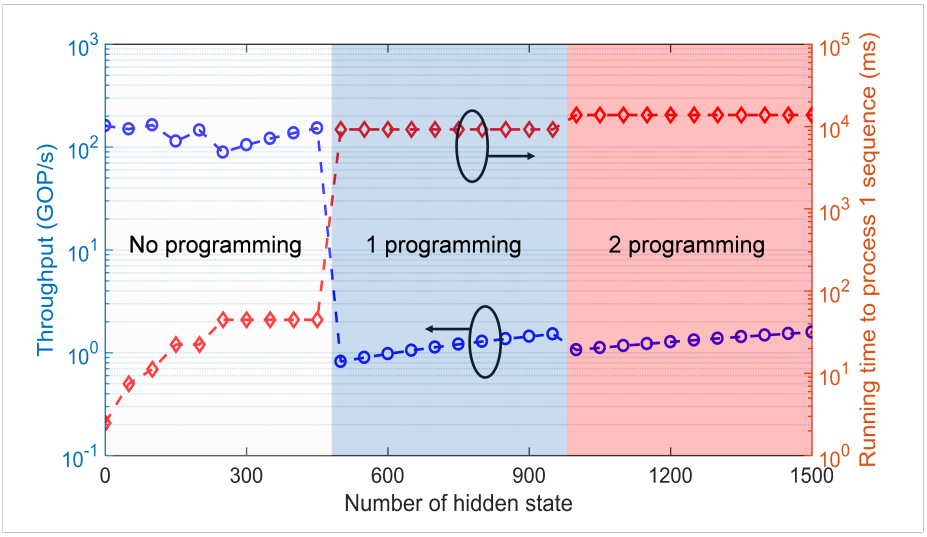


Figure 3.12: System performance with different number of hidden state.

HAR dataset, the performance of the basic RNN is not satisfactory (only 69%) while both LSTM and GRU achieve around 90% accuracy.

3.4.5 Handling large scale networks

Prior ReRAM based works do not consider the need for re-programming during inference. They assume that the network parameters are programmed into the processing arrays and never change. Even though PipeLayer [27] characterizes the device writing, it only considers the weight update during training. However, re-programming can be necessary, especially when the energy budget is constrained and the system capacity is not enough to hold all parameters simultaneously. It is critical to get a more comprehensive understanding about the system performance with re-programming considered.

Using LSTM as an example, we gradually increase the size of hidden states and evaluate the system throughput and running time with the system capacity unchanged (512 KB), shown in Fig. 15. When the hidden state number is small (< 500), the parameters of the network can be mapped to the system simultaneously. Moreover, multiple small LSTMs can be mapped to the system together, allowing processing several input sequences at the same time. On the other hand, if the hidden state size reaches a threshold when the param-

eter size is larger than the system capacity, re-programming occurs. We observe a drastic performance and speed drop due to re-programming. Further increasing the number of hidden state introduces more programming cycles along with more throughput and speed drop. Should aware that neural networks with different size of hidden state may have different performance even though they have same number of re-programming. For example, we consider LSTM with 600 and 900 hidden state (the parameter size is 628 KB and 928 KB, respectively). Since the system capacity is 512 KB, requiring 1 re-programming for both networks. Remember that the reading time is a constant, the inference time are same for these two networks. However, networks with more hidden states have more computation. Therefore, the performance (GOP/s) is different.

As mentioned earlier, we can increase the number of ReRAM processing arrays in a bank, or use multiple banks tiled together to increase the system capacity, and thus, avoid the re-programming issue. The energy consumption is proportional to the system scale. Another solution is storing more bit per cells. For example, if one device can store 4-bit, then the maximum capacity becomes 2 MB, 4 times larger than our original design (2-bit/cell). But this will introduce more computation error. The third approach is to increase the crossbar size. However, to drive larger array, the WL buffer size must be scaled up proportionally (more power consumption). Moreover, ADC with more bit precision is required for large crossbar array.

3.5 Summary

A RNN accelerator design based on ReRAM PIM architecture is presented in this research. The proposed architecture is suitable for various RNN computation including the basic RNN, LSTM, and GRU. The system throughput and energy efficiency with detailed circuits/devices characterization is measured. The computing efficiency of the proposed system achieves 79x improvements compared with GPU baseline on average. Further, the computing accuracy drop is insignificant when the read noise standard deviation is less

than 0.2. Lower bit-precision such as 8-bit can enhance the performance with insignificant accuracy loss. Re-programming during inference can significantly deteriorate the performance and should be minimized.

CHAPTER 4
FEFET BASED PIM ARCHITECTURE FOR DNN INFERENCE ACCELERATION

4.1 Introduction

Direct integration of computation and storage within a memory device can fundamentally eliminate the separation between compute and data, thereby enabling orders of magnitude higher energy-efficiency in data-intensive applications. There have been significant efforts in exploiting emerging non-volatile memory (NVM), in particular, resistive random-access memory (ReRAM), to perform in-memory computation [25, 26, 27, 31]. The key idea behind the ReRAM based accelerator is utilizing crossbar array to perform vector-matrix multiplication (VMM) using mixed-signal computation.

However, when examined closely from a circuit rather than micro-architecture perspective we note that designing a scalable architecture with ReRAM based in-memory computation remains challenging. First, a crossbar with many parallel ReRAM devices presents a low-impedance load to the input. This is fundamentally at odds with CMOS gates which are designed to drive high-impedance loads (i.e. capacitive loads). Although many prior works neglected this challenge, we show through circuit simulations that power-hungry analog drivers are necessary to ensure accurate computation in ReRAM crossbar. Second, as ReRAM has relatively low on state resistance ($1K\ \Omega$ to $100K\ \Omega$ for R_{on}) [77, 78, 79], the energy dissipation during VMM operation can be detrimental as all ReRAM devices in the crossbar simultaneously consume read current. Third, since the size of a crossbar is constrained by the peripherals, device re-programming and/or accumulating partial results from multiple crossbars are required to solve large problems. The high programming energy ($>$1 pJ/cell) [80, 81] in ReRAM as well as the in-efficient data movement can degrade

the computing efficiency. Finally, variability in ReRAM devices (or other NVM technologies) leads to inaccuracy.

This research argues that transforming the promise of in-memory computation to a scalable architecture requires a cross-cutting solutions connecting emerging device technologies, circuit techniques, and micro-architectural supports. Towards this end, this work proposes FERA, a ferroelectric FET (FeFET) based scalable architecture for data-intensive computing. FeFET has similar structure with a normal MOSFET, except it has a ferroelectric layer inside the gate. The polarization of the ferroelectric layer can be switched, thereby, the transistor threshold voltage can be tuned. The development of FeFET has made tremendous progress in recent years with demonstrations from commercial foundries [82, 83, 84].

FERA uses FeFET technology to exploit fine-grain concurrency in data-intensive computation in a manner similar to ReRAM engines, but instigates major gains in energy-efficiency by co-designing the peripheral circuits and interconnection network. This enables energy efficient scalability of the VMM compute engines supported by an optimized data partitioning and data flow based execution model.

FERA is built on three core concepts, namely, (i) leverage unique properties of FeFET using co-design of technology and circuit to improve energy-efficiency; (ii) enable scalable micro-architecture by connecting multiple VMM PEs using a hierarchical Network-on-chip (NoC) with in-router processing and optimized data partitioning and mapping; and (iii) exploit dynamic fixed-point data in VMM computation to reduce the effect of device variability.

FeFET based VMM engine: As a three-terminal transistor device, FeFET provides a high-impedance gate terminal and very high on/off ratio, thanks to its steep switching slope [85]. A crossbar architecture is proposed where each cell performs 1-bit multiplication (i.e. AND logic). The high on/off ratio of FeFET ensures high computing accuracy, while low read current ($\sim$ 1 nA/cell) and programming energy ($\sim$ 1 fJ/cell) reduces

crossbar energy [82]. The unique characteristics of FeFET is leveraged to eliminate analog peripherals of ReRAM, and design lightweight memory-like digital peripherals. To be more specific: first, it shows that low-power digital drivers are sufficient to drive the high-impedance gate of the three-terminal FeFET. Second, the power-hungry ADC is replaced with a pre-charge/discharge circuit and sense amplifier (SA) to realize the function of time-to-digital conversion (TDC). Since the same peripherals of normal memory is employed, FeFET based VMM engine can be easily reconfigured for data storage.

Micro-architecture for efficient data communication: this work presents a communication fabric connecting the VMM engines using a H-tree hierarchical network-on-chip (H-NoC). Routers with embedded logic is utilized to process the partial results within the NoC. The proposed H-NoC is coupled with optimized partitioning of matrices and data flow to enables efficient scaling of FERA architecture using many VMM engines. The result is a VMM processing platform that is designed to be scaled architecturally. The circuit and device technologies are matched to enable the energy efficient scaling to larger numbers of arrays.

Dynamic fixed-point data format for robust computation: We employ dynamic fixed-point data format in FERA, instead of fixed point as in prior works, to improve computing accuracy under device variations. FERA uses a data flow based execution model that interacts with H-NoC to enable accumulation of partial results. Currently, FERA uses layer-by-layer programming and execution of DNN. The programming infrastructure includes a front-end application software, such as Tensorflow, to extract the DNN parameters, determine the VMM configuration, and a software-hardware interface to schedule the data flow for a specific DNN layer. Once the data flow is schedule, FERA operates autonomously to load the inputs, start execution, collect back computing results, and if required, perform re-programming of the VMM engines.

The architectural performance of FERA is evaluated using a custom cycle-level simulator for benchmark convolutional neural networks (CNNs). The impact of device vari-

ations (bit-flip error and stochastic Gaussian noise) to the computing accuracy is evaluated. The performance, power, and accuracy analysis is driven by experimentally calibrated FeFET models, and detail semi-custom design in 28nm CMOS including full-custom (schematic/layout) design of VMM crossbar and peripherals, coupled with synthesized designs for digital components such as activation&pooling unit, H-NoC with in-router processing, data flow controller, etc.

Simulation in 28nm CMOS indicates that, FERA reduces average power while maintaining high accuracy under variation. The FeFET based crossbar achieves 1000x lower energy dissipation than ReRAM crossbar, thanks to reduced read/write energy. However, detailed circuit simulations show that simply replacing ReRAM crossbar with FeFET crossbar will lead to only 1.2X reduction in power, as the power is dominated by the peripheral circuits rather than the device itself. A VMM engine in FERA with optimized FeFET crossbar size and optimized peripheral shows 6.3X lower power than a ReRAM VMM engine of same size. Moreover, an efficiency optimization is presented that couples design of the VMM engines with the H-NoC to optimize the crossbar size and maximize computing efficiency (GOPS/s/W) of the FERA system. Overall, FERA shows $254\times$ and $9.7\times$ gain in computing efficiency compared with GPU and ReRAM based design, respectively.

4.2 FeFET basics

Ferroelectric FET is a transistor in which the ferroelectric oxide layer is included in the gate dielectric stack. A ferroelectric oxide is an insulator which exhibits a spontaneous electric polarization in the absence of electric field. The direction of the polarization can be switched by applying a voltage larger than the coercive voltage. In a FeFET, the bistability of the ferroelectric polarization gets coupled to the channel current leading to the non-volatile memory properties in a transistor structure. A FeFET can have programmable multiple threshold voltage states corresponding to the partially switched polarization states of the ferroelectric. Even though researched for several decades, traditional ferroelectric

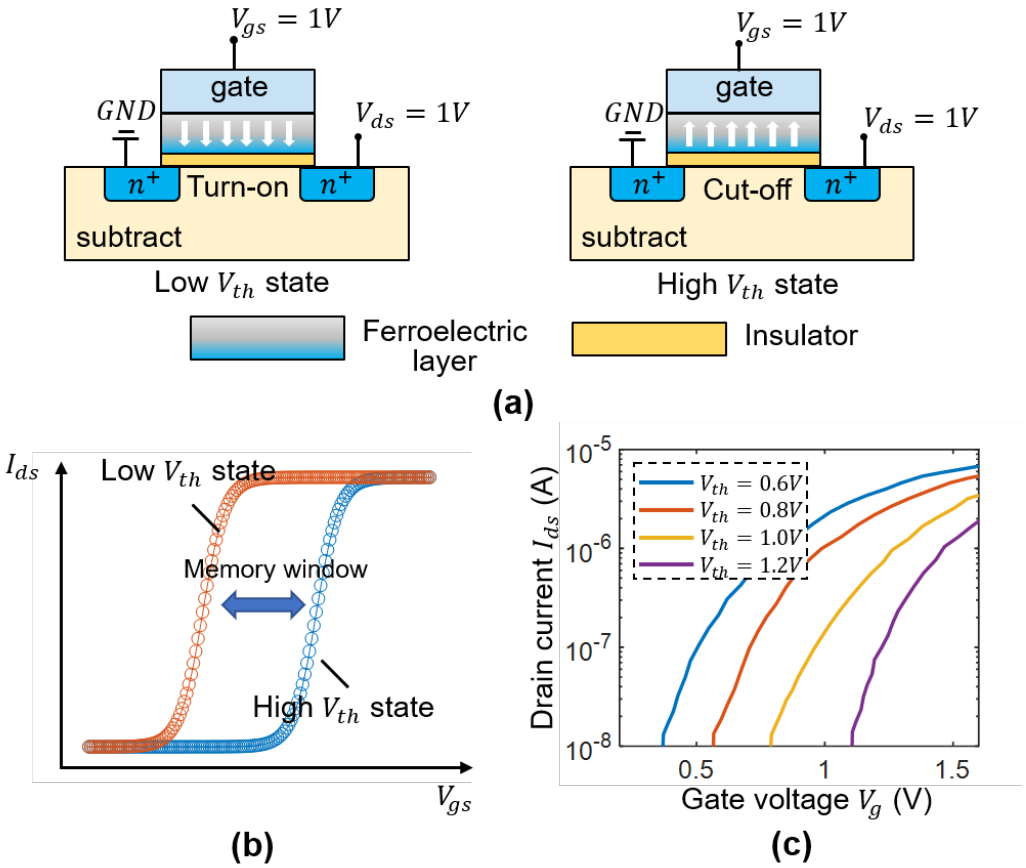


Figure 4.1: FeFET basics: (a) A typical FeFET structure where the ferroelectric layer is sand-witched inside the gate dielectric. (b) FeFET hysteresis loop with binary state encoded. (c) Gradual switching of the ferroelectric layer and corresponding I-V characterization [90].

materials like $Pb(Zr,Ti)O_3$ (PZT), $SrBi_2Ta_2O_9$ (SBTO) still have fundamental challenges such as CMOS incompatibility, data retention and scalability [86]. Thanks to the recent discovery of ferro-electricity of silicon doped hafnium oxide (Si:HfO_2), the HfO_2 thin file based FeFET is transferred to the mainstream CMOS platform due to its CMOS compatibility [82, 87, 88, 89]. Therefore, in this work, we focus on a subset of FeFET device, which use HfO_2 thin film to realize the ferro-electricity.

The ferro-electricity originates from the fact that, in the crystal unit cell, the central ion resides in two stable off-centered positions, shifting either up or down according to the polarity of an externally applied electric field [91]. Depending on the position of this ion, a

Table 4.1: Comparison between FeFET and ReRAM. (Date in parentheses are the best reported results from literature.)

Device characterization	**FeFET [82]**	**ReRAM [78]**
Write Endurance	$10^5(10^9)$	$10^6(10^{10})$
Date Retention	$>$ 10 years	$>$ 10 years
Write speed	500 ns (10 ns)*	50 ns (10 ns)
Write energy	$\sim$ 1 fJ	$\sim$ 5 pJ (1 pJ)
On/off Ratio	$> 10^3$	< 10 (10^3)
Area	4 F^2	4 F^2

electric polarization can be created which can control the threshold voltage and, therefore, the channel conductance of FeFET transistor. As shown in the bottom of Figure 4.1 (a), when the polarization is pointing downwards, channel is in inversion, bringing the transistor into the 'ON' state ($i.e.$ low V_{th} state). Similarly, if the polarization is pointing upwards, channel is in accumulation which gives the transistor 'OFF' state ($i.e.$ high V_{th} state). Direction of polarization is non-volatile and can be programmed by the external programming voltage. Figure 4.1 (a) shows the FeFET hysteresis loop with binary state encoded. Moreover, gradual switching of the ferroelectric layer (i.e. multi-level of threshold voltages and channel conductance) has been demonstrated in several recent works [90, 92, 93]. Figure 4.1 (c) presents the experimental data showing 4 different levels of transistor threshold voltages [90].

Hafnium oxide based FeFET has attracted intensive research interests since its discovery [82, 89, 94, 95]. It has already been demonstrated that Hafnium oxide FeFET has good temperature stability, writing endurance, data retention and switching speed/energy which make FeFET now comparable or even better than other non-volatile memory candidates such as ReRAM (Table 4.1). The ultra-low writing energy due to the unique electrical field effect switching mechanism is the most prominent feature which distinguish FeFET from other emerging technologies.

Besides utilizing FeFET as non-volatile memory [86, 82], there have been a few recent works exploring FeFET based logic design[96], neuromorphic computing [97, 98, 99, 100]

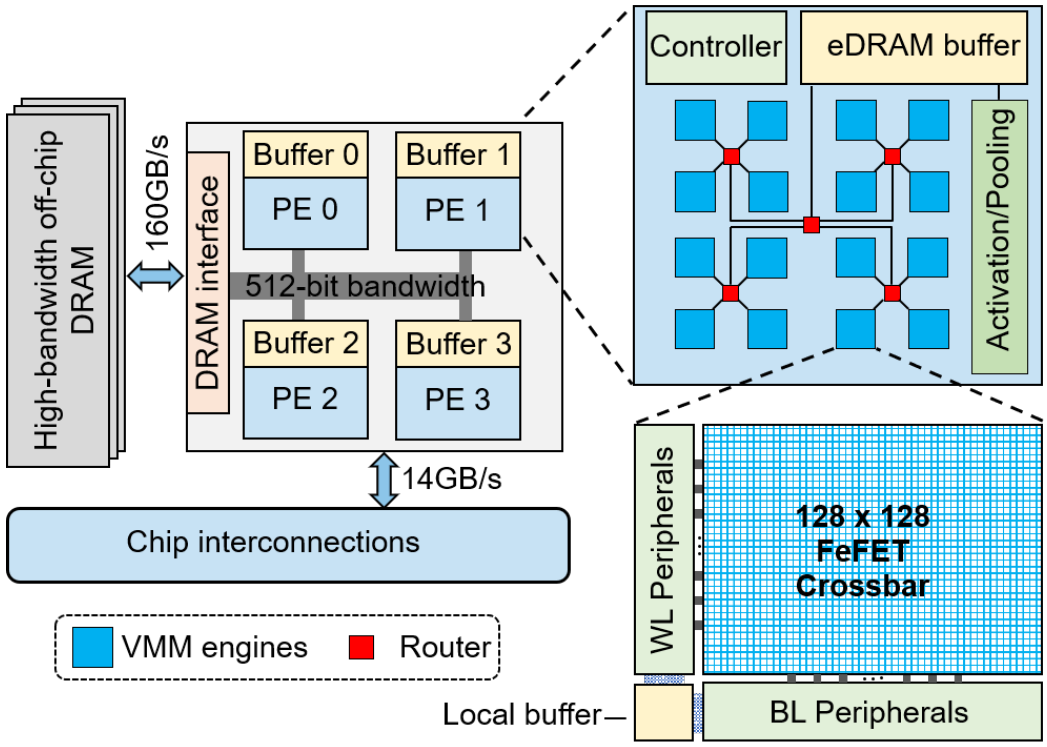
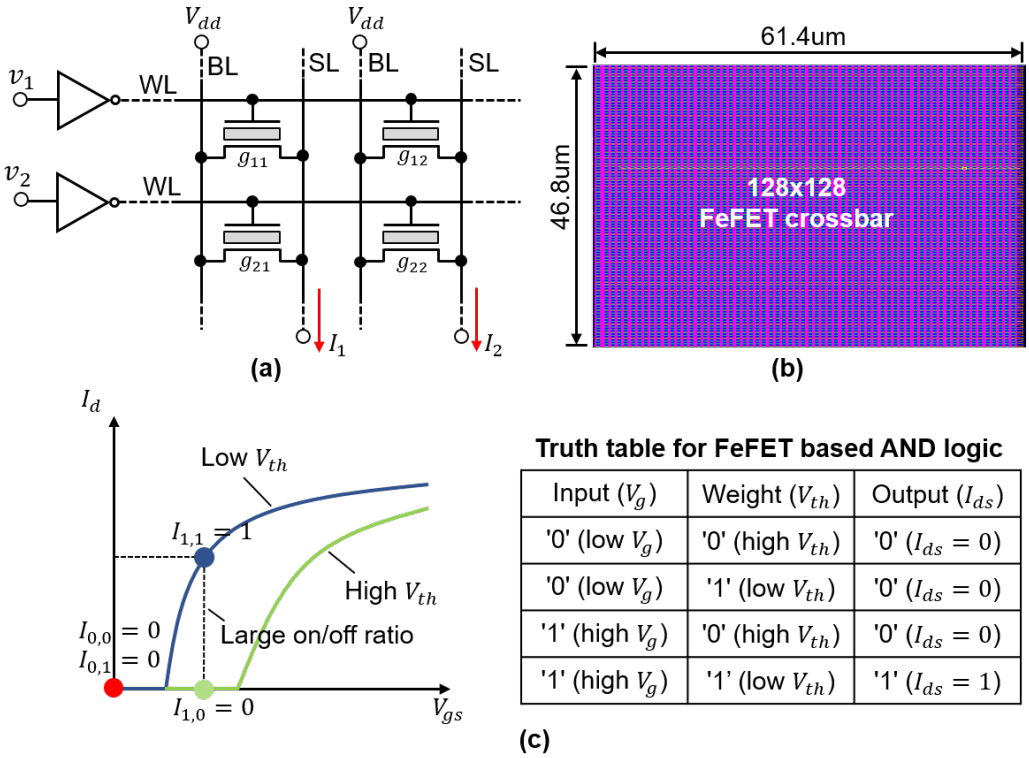


Figure 4.2: Overview of FERA system architecture.

and non-volatile computing [101]. However, most prior works focus on device modeling and lack of system/architecture level design. In this work, we propose a scalable architecture using FeFET crossbar as VMM engine to accelerate data-intensive applications.

4.3 System architecture

Figure 4.2 shows the overview of FERA architecture consisting of multiple parallel processing engines (PE) connected to an off-chip memory. Inside each PE, there are a set of interconnected VMM engines. Our current design assumes there are 256 VMM engines in each PE (Figure 4.2 only shows 16 VMM engines for simplicity). A PE also contains one global buffer to store the temporary input/output data, and an activation/pooling unit to handle the activation function and pooling operations. A dedicated H-tree like on-chip network (we call it H-NoC) is utilized to transmit data between buffer and VMM engines. At the bottom level, each VMM engine consists of a FeFET crossbar with 128×128 devices,



Input (V_g)	Weight (V_{th})	Output (I_{ds})
'0' (low V_g)	'0' (high V_{th})	'0' ($I_{ds} = 0$)
'0' (low V_g)	'1' (low V_{th})	'0' ($I_{ds} = 0$)
'1' (high V_g)	'0' (high V_{th})	'0' ($I_{ds} = 0$)
'1' (high V_g)	'1' (low V_{th})	'1' ($I_{ds} = 1$)

Figure 4.3: (a) Configuration of FeFET crossbar. (b) Layout view of a 128×128 crossbar. (c) FeFET based 1-bit multiplication (i.e. AND logic).

WL/BL peripherals, and a small local buffer.

4.3.1 FeFET based VMM engine

FeFET for 1-bit multiplication

Figure 4.3 (a) shows the configuration of FeFET crossbar where gate, drain, source of the transistor are connected to WL, BL and source line (SL), respectively. Figure 4.3 (b) shows the corresponding layout view of a 128×128 crossbar. In computing mode, DNN weights are stored as transistor channel conductance (i.e. threshold voltage) and input vectors are used to drive WLs (i.e. transistor gate).

For ReRAM, the read current is multiplication of applied voltage (DNN's input) and device conductance (DNN's weight) ($I = V \times g$). However, FeFET is a field effect device where drain current (I_{ds}) depends on the difference between the gate voltage (V_{gs}, repre-

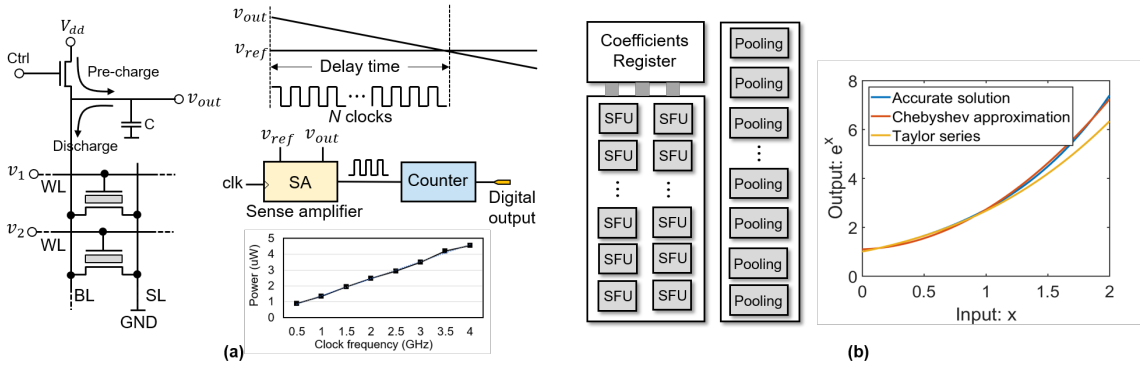


Figure 4.4: (a) Pre-charge/discharge based reading scheme and the SA based TDC design. (b) Activation/Pooling unit. The activation function unit is implemented based on Chebyshev approximation which can achieve better accuracy than Taylor series approximation.

sents input) and the threshold voltage (V_{th}, represents weight). Hence, directly performing the multiplication of input and weight is not possible in FeFET.

We propose to perform 1-bit multiplication (i.e. AND logic) in the cell (Figure **??**(c)). One-bit of weight is encoded as high V_{th} or low V_{th}, representing either 0 or 1, respectively; similarly, 1 bit of input vector can be encoded as high or low WL voltage (i.e. V_{gs}). When the input bit is 0 (i.e. low V_{gs}), the current is always 0 with either high V_{th} or low V_{th} since the transistor is turned off (red dot in Figure 4.3 (c)). On the other hand, if the input bit is 1 (i.e. high V_{gs}), the transistor is still off when V_{th} is high (green dot in Figure 4.3 (c)) but turned on when V_{th} is low (blue dot in Figure **??**(c)). The large on/off ratio of FeFET, thanks to its steep sub-threshold slope (¡60 mV/dec) [85], creates large difference between the output '1' current and output '0' current.

Peripherals

One advantage of our design is that now the WL connects to transistor's gate which is a capacitive load. Therefore, there is no word line voltage drop and input perturbation issue as in the ReRAM scenario. This allows us to use traditional digital CMOS drivers (inverter) for WL, reducing peripheral power dissipation without sacrificing accuracy.

One challenge of ReRAM based VMM engine is the power/area overhead of the ADC.

For example, in ISAAC [26], a 8-bit ADC is employed to covert the summed current to digital value, consuming 49% power and 23% area. PipeLayer [27] replaces the ADC with an integration & fire circuit based design. However, the comparator (normally designed with an Op-amp) still consumes significant amount of power. Distinguished from prior works, inspired by the reading scheme of the conventional memory, FERA employs a pre-charge/discharge approach as shown in Figure 4.4 (a). At the beginning, the BL is pre-charged to the supply voltage (V_{dd}). Then, depending on how many transistors in the same column are turned on, the BL voltage (V_{BL}) droping slop varies. We utilize a sense-amplifier (SA) to sample the difference between the reference voltage (V_{ref}) and V_{BL}, the output from SA is a series of pulse, used as the input of a pulse counter. Basically, with a simple SA and counter, we realize the time to digital converting (TDC). Our SPICE simulation indicates that, for a 128×128 array, the 128 TDCs power is $1.3\,uW$, $2\times$ less than the ADC based approach in ISAAC [27]. Note, there is a power and latency trade-off for the SA based TDC design, as shown in Figure 4.4 (a). With higher clock frequency, the time-to-digital converting latency is reduced, but the power consumption for SA and counter will increase.

Activation/Pooling unit

We implemented an activation/pooling unit to handle the computation of activation function and pooling operation, shown in Figure 4.4 (b). A Chebyshev approximation based special function unit (SFU) is design to solve the activation function such as sigmoid (σ), or hyperbolic tangent ($tanh$). Chebyshev polynomial is very close to the optimal polynomial and can achieve better accuracy than Taylor expansion with same number of coefficients [73] as illustrated in Figure 4.4 (b) for approximating exponential function as a 3rd order polynomial. The coefficients for SFU are first calculated and pre-loaded into the local register file. SFU accesses the register file and calculates the non-linear function. In prior ReRAM accelerator works, circuits are designed to solve a specific activation function [**2016isaac**,

25]. Compared with them, the SFU based approach is advantageous since it has more flexibility, can be easily reconfigured (i.e. load different coefficients) for different functions. The output of SFU is routed to pooling logic or sent back to global memory, depending on the layer type.

On-chip memory

A mix of SRAM and eDRAM are employed as the on-chip memory. The small (128 Byte) local buffer in each VMM engine to store temporary input and output data is implemented using SRAM. Each PE also contains a global buffer implemented using eDRAM. The global buffer receives input data from off-chip DRAM and collect computing results from FeFET arrays. The size of global buffer is 16 KB. In total, there are 192 KB on-chip memory (local buffer + global buffer) in a FERA chip.

Reconfigurability

The proposed FeFET crossbar configuration is similar with the AND memory architecture [82, 84]. Moreover, both the WL/BL peripherals and the pre-charge/discharge of BL are same with a conventional memory, except an additional pulse counter. Therefore, we can reconfigure the FERA either in computing mode (i.e. VMM engine) or in traditional memory mode (i.e. bit-storage). The only difference is that all WLs are activated simultaneously when FeFET crossbar is used for computing, while only one WL is enabled when used for memory.

4.3.2 Micro-architectural support

In this subsection we discuss the micro-architectural support including data partitioning and mapping, the communication architecture, and how they are integrated to design a scalable system.

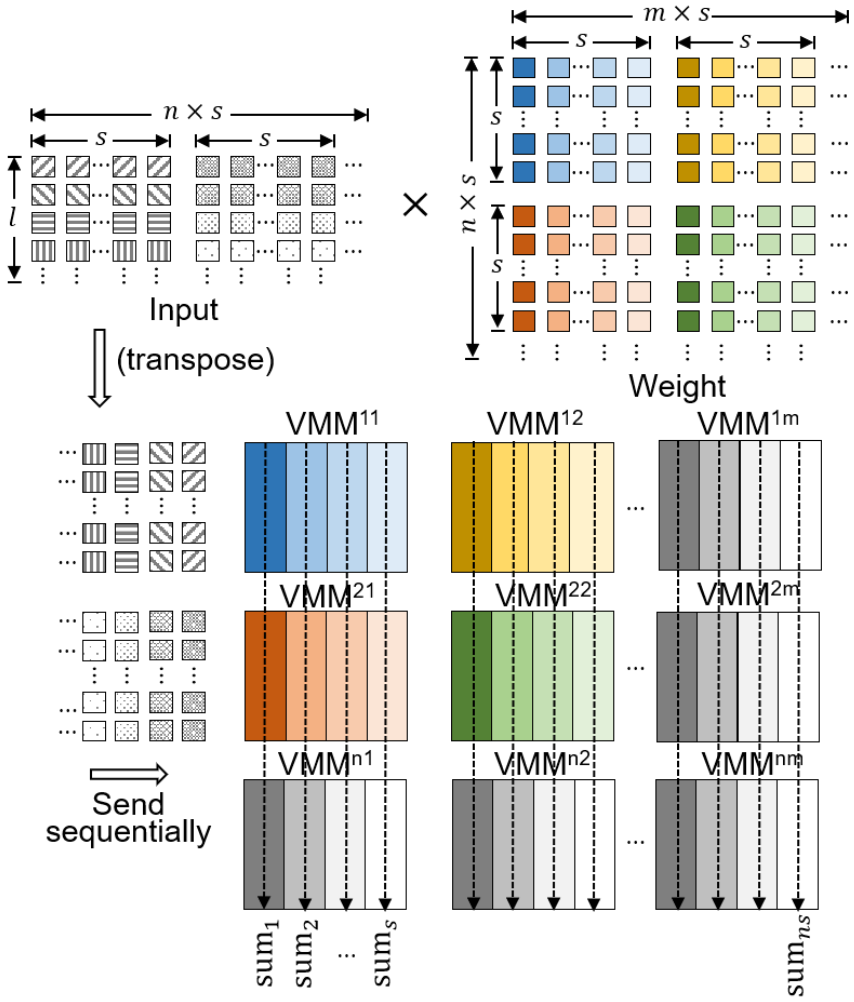


Figure 4.5: Matrix partition and mapping to multiple VMM engines.

Data Partitioning and mapping

Figure 4.5 illustrates how to partition a single matrix operation across multiple VMM engines. Assuming the crossbar inside the VMM engine is $s \times s$, the weight matrix is then partitioned into several small segments with the granularity of $s \times s$. Each partition is then assigned (programmed) to a VMM engine, in total, $n \times m$ VMM engine will be used (the definition for n and m are shown in Figure 4.5). The input matrix is first transposed and sequentially fed into the VMM engines. Each column is partitioned and applied to the input of VMM engines. Note, each input segment is shared across multiple VMM engines in the row direction (e.g. VMM^{11}, VMM^{12}, till VMM^{1m} in Figure 4.5). However, partial results

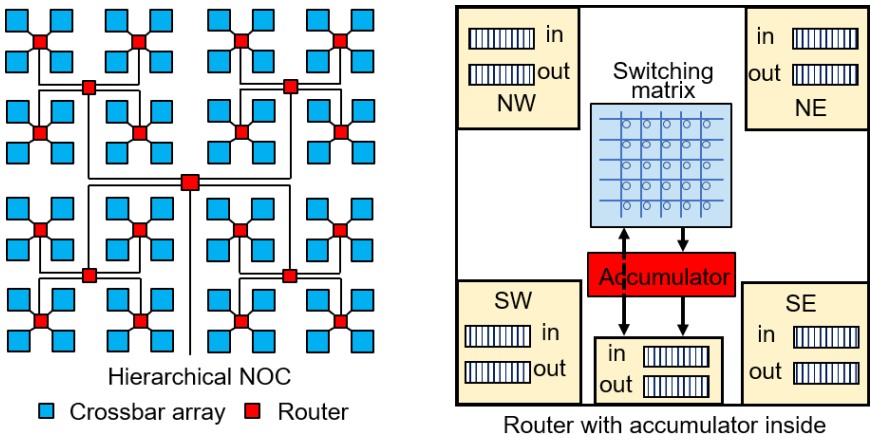


Figure 4.6: (a) Hierarchical network-on-chip. (b) Router design with accumulator integrated.

generated from the VMM engines in the same column should be summed together (e.g. VMM^{11}, VMM^{21}, till VMM^{n1} in Figure 4.5).

VMM organization and H-NoC design

H-NoC is specifically designed to address the discrepancy between row-wise input sharing and column-wise output summation. Figure 4.6 (a) presents the organization of VMM engines and their connection (via H-NoC). At the bottom level, 4 VMM engines share a router. Then, 4 such routers are connected to a router in the higher level. Considering 256 VMM engines in a PE, there are $256/4 = 64$, 16, 4, and 1 routers from the bottom level to the top level, respectively (Figure 4.6 (a) only shows 3 levels). In total, within 1 PE, we have 85 routers distributed in 4 levels. Figure 4.6 (b) shows the router design, containing five input/output ports and corresponding I/O buffers. Four ports are connected to router's neighborhoods, either the nearest VMM engines or routers in the lower level. The 5th ports connects to the router in the upper level (router on the top level directly communicate with global buffer). A 5×5 switching matrix is equipped to route input/output ports and the routing is based on store-and-forward (SAF) approach. Distinguished from conventional router designs, we insert a simple accumulator inside the router to enable on-the-fly partial results summation.

Operation of H-NoC

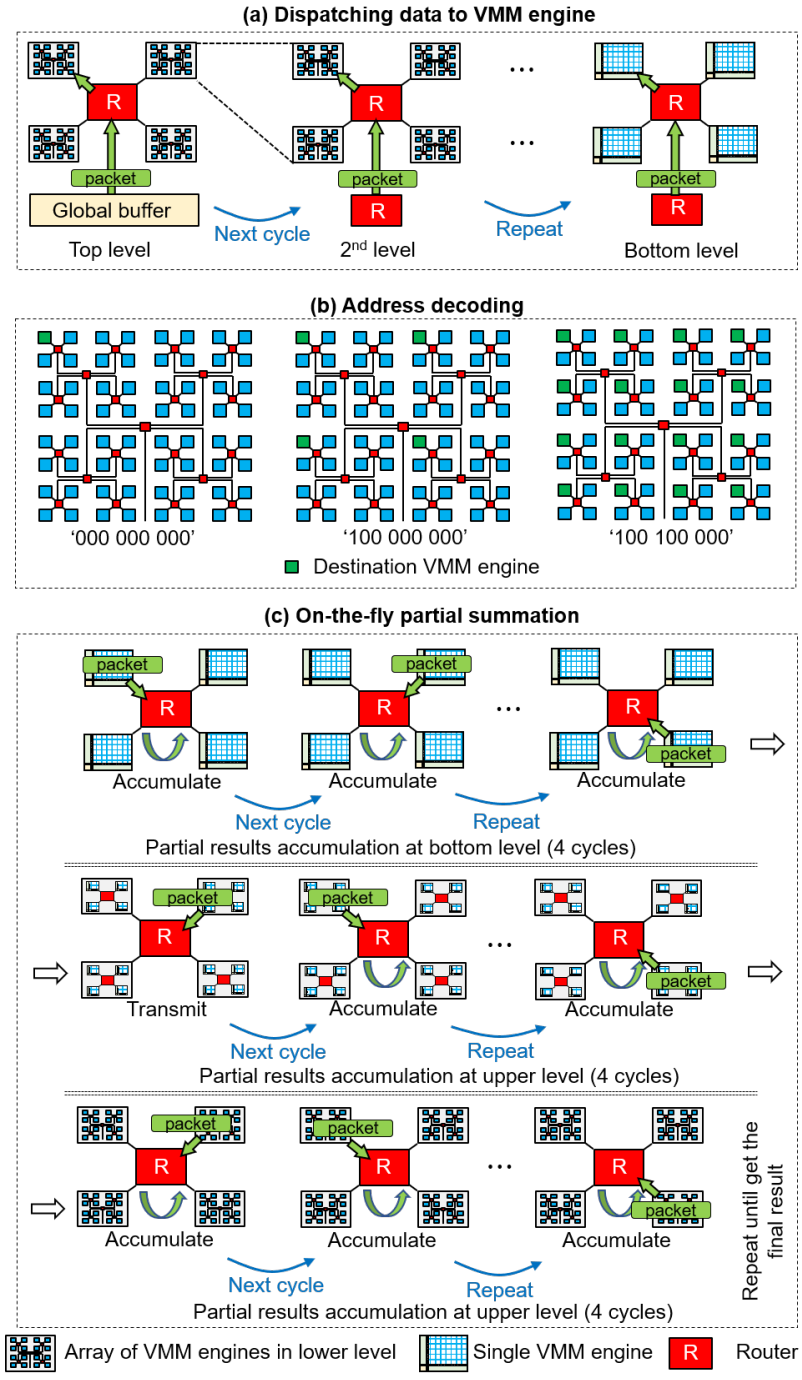


Figure 4.7: (a) Dispatching data from global buffer to individual VMM engines. (b) Partial results accumulation in a layer-by-layer fashion.

The H-NoC has three key operations: (i) *data dispatching*: input data are sent from

global buffer to each VMM engine; (ii) *data collecting*: VMM outputs are pushed back to the global buffer; and (iii) *partial data summation*: on-the-fly summation of partial results from multiple VMM engines within the H-NoC .

Figure 4.7 describes how data are transmitted inside H-NoC for the aforementioned three scenarios. Figure 4.7(a) illustrates the data flow for dispatching. When the global buffer issue a packet, the root router first decodes the address and determines which output port the packet should go. Then, at the next clock cycle, router in the lower level receives the packet, decodes, and sends. This process is repeated until the packet arriving the designated VMM engine. The address is contained in the packet (packet head). The root router decodes the first 3-bit and sends packet to its NW, NE, SW, SE output ports when the address is '000', '001','010','011', respectively. However, if the address is '100', the packet will be routed to all the output ports, simultaneously. The router at the lower level decodes the 4-6 bits of the address with the same logic, etc. The data fetching mode works in the same way except the data flow is reversed.

Figure 4.7 (b) shows the operation of the H-NoC working at partial data summation mode. The partial result is generated from each VMM engine, which are accumulated as data flows from the bottom layers to the top layer. At the bottom level, a router receives four partial results from the connected VMM engines, accumulates the data in 4 cycles, then it sends the accumulated result to the upper level router. The routers located at the upper level repeat the same process. Once all the partial results are added, the result is collected to the memory. As routers in the same level are working in parallel, the worst-case latency is limited to $4 \times \ number\,of\,levels$.

Figure 4.8 shows how the different weight partitions are mapped to the hierarchically connected VMM engines. We map 4 segments of the weight matrices in a single column (represented with small square in same color at the left side of Figure 4.8 (a)) to a single leaf router node (right side of Figure 4.8 (a)) in the H-NoC tree to leverage in-router accumulation of partial summations across column. Note, the weight matrix partitions sharing

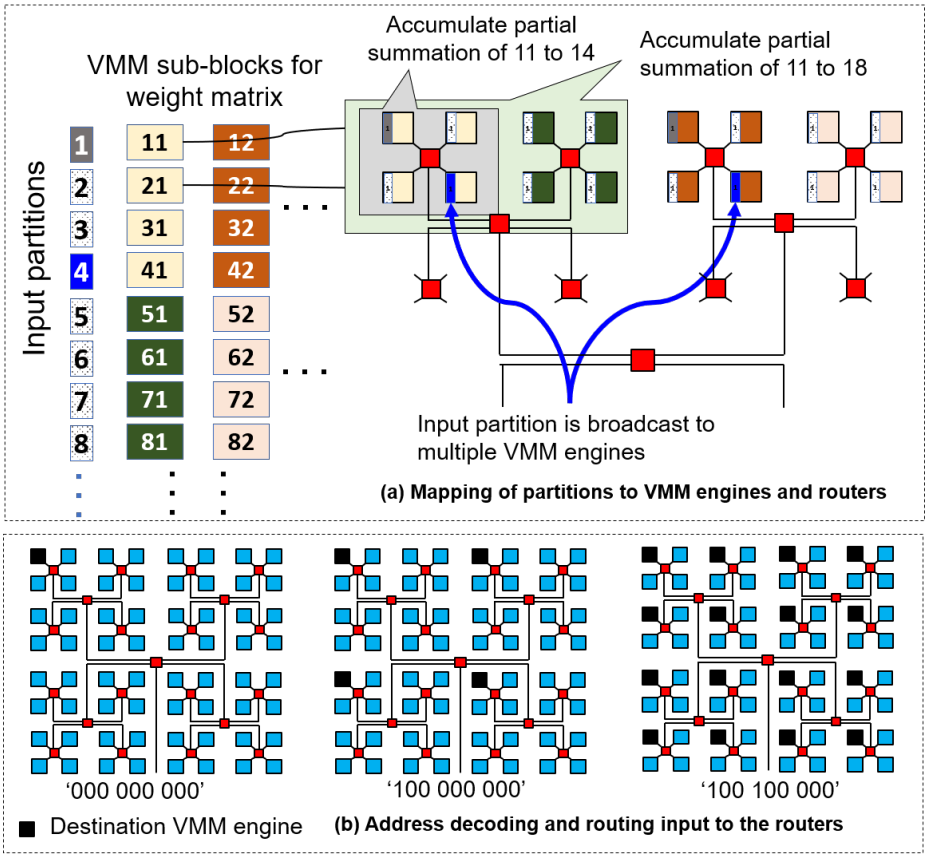


Figure 4.8: Mapping of VMM engines to H-NoC and broadcast of input partitions

same inputs (partitions in the same row) are now mapped to different branch of the H-NoC tree. As we have discussed, H-NoC supports broadcasting a single input vector to multiple VMM engines in different branches, as shown in Figure 4.8 (b). Utilizing this address decoding mechanism, the packet can be transmitted in a one-to-one fashion or broadcasted in any level, realizing the input sharing (i.e. broadcast the same input vector to multiple VMM engines without re-sending).

Note that the VMM operation and input transfer are in pipeline, ensuring high transmission rate and clock frequency. Also, the proposed H-NoC is naturally deadlock free since the routing only happens in the up-down directions.

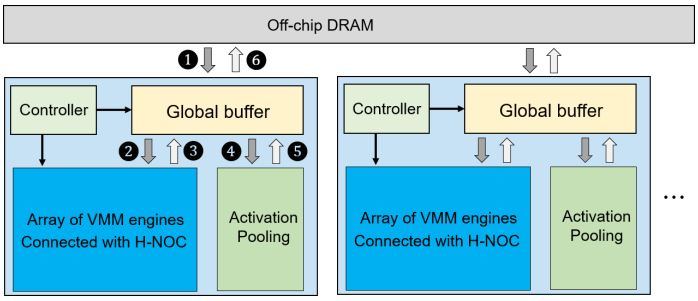


Figure 4.9: Chip-level execution model.

4.3.3 Execution Model

Figure 4.9 illustrates the chip-level execution model which contains 6 steps. 182 The controller inside PE asks the memory interface to load data from off-chip memory and stores in the global buffer. Benefiting from HMC's high bandwidth, each PE has its individual link which comprise of 16 input lanes and 16 output lanes with the maximum aggregate bandwidth of 80 GB/s (the total bandwidth of 4 links are 240 GB/s) [102]. Therefore, 4 PEs are working in parallel. 183 The data are then dispatched to VMM engines via H-NoC for computation.

184 After the computing is done, partial results are first summed up on-the-fly and then collected back to the global buffer. At step 185 and 186, the output from VMM engine arrays is fed into the activation/pooling unit (down sampling is omitted if there is no pooling layer). 187 The final result is sent back to off-chip memory.

4.3.4 Supporting dynamic fixed point

NVM based computation is error-prone due to the device variation [78, 82]. FERA employs dynamic fixed-point representation to enhance the error tolerance under device variation. The dynamic fixed-point format allows several decimal points instead of single global one in the traditional fixed-point format [103]. With dynamic fixed-point format, we can maximize the bit utilization by minimizing the length of integer part. As shown in Figure 4.10, with a 6-bit number (3-bit integer) stored in NVM array, the read out data would be heavily

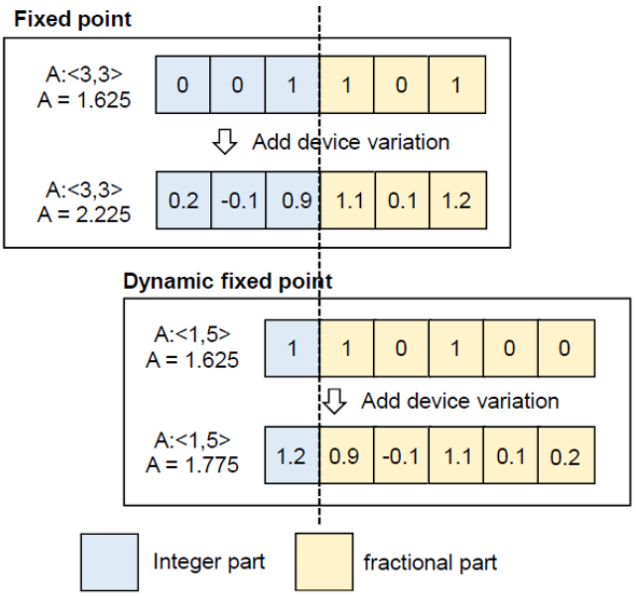


Figure 4.10: Using dynamic fixed-point data format to enhance the error tolerance for device variation.

disturbed by the device variation from MSB. On the other hand, with dynamic fixed-point, the demical point position can be dynamically changed, enabling the maximum bit utilization (MSB is always the first valid bit), as shown in Figure 4.10. Therefore, the error introduced by device variation is minimized.

We should note that several recent works have demonstrated that dynamic fixed-point representation is beneficial to speed up the training of machine learning applications [104, 105]. However, rather than accelerating computing, here we exploit the dynamic fixed point as a technique to reduce the impact of device variation.

4.3.5 Programming model

A programming model (as well as the corresponding FERA simulator) is developed to let the user easily configure FERA to handle different DNN applications, shown in Figure 4.11. The front end contains a user-defined network configuration file (*DNN.cfg*) and the corresponding trained data (*Data.mat*) exported from machine learning packages such as Tensorflow and Caffe [76, 106]. The second stage is the software-hardware interface, per-

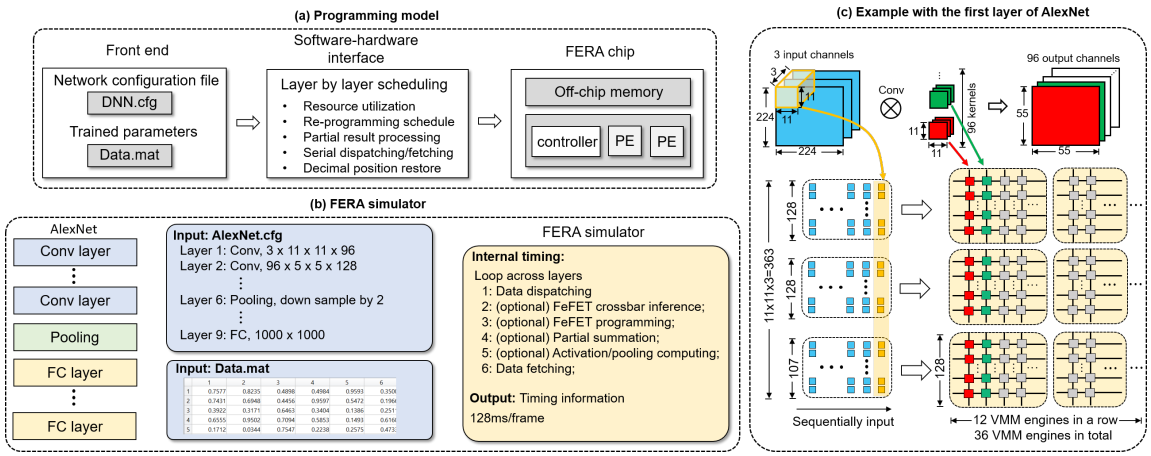


Figure 4.11: (a, c) Three-stages programming model with an example showing the mapping procedure. (b) FERA simulator.

forming layer-by-layer scheduling. The scheduler generates a detailed (cycle-level) agenda for data dispatching/collecting, partial results processing, and crossbar re-programming (if necessary). Moreover, the scheduler also determines the decimal points for each layer (based on the parameters range). Then, the schedule is sent to the on-chip controller for hardware execution. Note, after performing computation for each layer, shift and rouding are required to restore the decimal point position before sending the data back to memory.

The first layer of AlexNet is used as an example to illustrate how the programming model works, Figure 4.11 (c). The scheduler first reads the configuration file and gets the layer specification (The kernel size is $3 \times 11 \times 11 \times 96$, corresponding to the number of input channels, kernel width, kernel length, output channels, respectively.) Based on it, the scheduler then calculates how many VMM engines are required to perform the computing (Each set of the convolution kernel (i.e $3 \times 11 \times 11$) contains 363 weight parameters, and thus, $*363/128 = 3$ crossbar arrays to perform the dot-production for one kernel. Since there are 96 kernels, in total we need $3 \times (96/8) = 36$ VMM engines. Note that 128 devices in a row can hold 8 16-bit numbers). Then, the scheduler will generates the schedules for re-programming (loading weight matrix to VMM engine), data dispatching/collecting (sequentially dispatch/collect input data to/from VMM engine), and activation/pooling op-

erations.

At last, the performance of FERA is simulated using a custom cycle-level simulator which is directly built on the programming model (software-hardware interface). As shown in Figure 4.11 (b), with the configuration and trained data of AlexNet [1] as input, FERA simulator calls the scheduler to give the number of cycles for each operations as well as resource utilization (how many VMM engines are involved). The cycle time is estimated based on detail circuit analysis/simulation as discussed below. Ultimately, the application level performance evaluation is based on the coupling of timing analyses from cycle-accurate simulator and power modeling from circuit-level simulation.

4.4 Results

4.4.1 System implementation

Table 4.2: Power and area of FERA chip

Component	**Power (mW)**	**Area (um^2)**	**Number**
WL peripherals	0.0001	0.38	128
BL peripherals	0.011	14.8	128
Local buffer (128 Bytes)	0.04	972.6	1
FeFET crossbar	0.00098	2873.5	1
Array total	1.46	5789.1	256
Activation/pooling	19.2	165376	1
H-NoC	170	1616700	1
Global buffer (16 KB)	5.2	21000	1
controller	0.48	940.3	1
PE total	0.568 W	3.29 mm^2	4
Chip total	2.274 W	13.14 mm^2	1

Since FeFET has the same planar topology with normal MOSFET, we layout crossbars with different size using Cadence Virtuoso to model area and wire parasitic. Figure 4.3(b) shows the layout of a crossbar containing 128×128 transistors. The SPICE simulation is then performed in 28nm CMOS technology using extracted netlist of the crossbar together

with the proposed WL/BL peripherals to estimate power and delay of the VMM engine. Thereafter, the SPICE simulation of the mixed-signal VMM engine is coupled with synthesized (using 28nm CMOS technology) digital blocks in FERA (such as shift & add unit, activation/pooling unit, H-NoC, and controller) to form a completed chip-level modeling. Synopsys Design Compiler and PrimeTime are used to model the power and area of the synthesized components. The on-chip memory is modeled with CACTI [**cacti**]. Table 4.2 summarizes the power and area of each block of FERA. The total chip power is 2.27 W, and the chip area is 13.14 mm^2.

Table 4.3: Benchmark DNNs configurations

CNNs	Number of layers	Parameter size (MB)	GOPs
AlexNet [1]	8	243.8	0.7
GoogleNet [107]	22	6.9	1.5
VGG-16 [2]	16	553.4	14.5
VGG-19 [2]	19	574.6	18.7

4.4.2 Experiment setup

Benchmark. The benchmark comprises 4 different well-known CNNs/DNNs, namely, AlexNet [1], GoolgeNet [107], VGG-16, and VGG-19 [2]. The detailed configurations (number of layers, parameter size, and number of operations) for the benchmark DNNs are listed in Table 4.3. We evaluate the benchmarks performance with a large and sophisticated dataset, ImageNet [23]

GPU baseline. We evaluate our benchmarks with Caffe deep learning framework [106] running on a state-of-the-art NVIDIA GTX 1080Ti GPU.

4.4.3 Computing efficiency optimization

For a given total FeFET memory capacity (on-chip resources), the design space is explored to maximize the efficiency by co-designing the FeFET crossbar, VMM engine, and H-NoC. As illustrated in Figure 4.12, the computing efficiency, speed, and power under

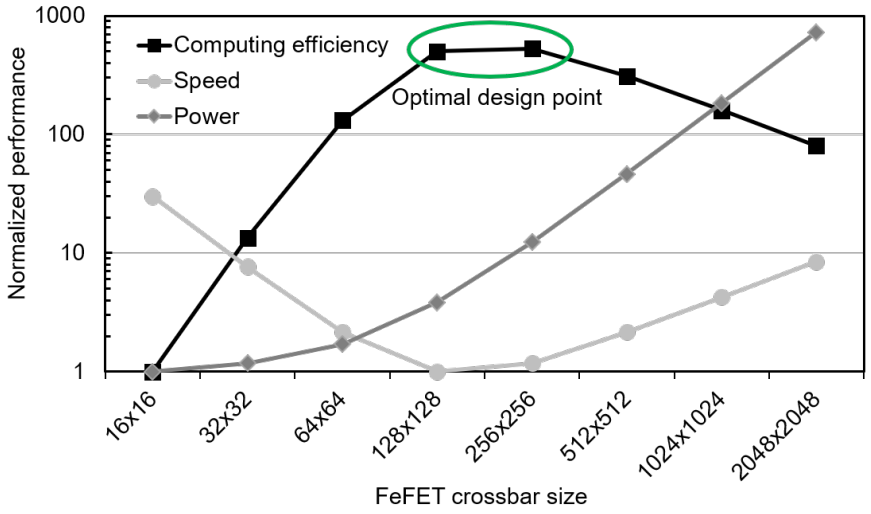


Figure 4.12: Computing efficiency optimization.

varying FeFET crossbar size (as well as the corresponding modified peripherals and H-NoC organization) are evaluated. Should note that all data are normalized based on the minimum value of the explore region. The smaller crossbar size reduces power within a VMM engine, but requires more clock cycles for data transmission. On the other hand, very large crossbar is also prohibitive due to the large RC delay and unmanageable power consumption at peripherals. The optimal design point is observed to be the crossbar size of 128×128 or 256×256. To have an apple-to-apple comparison with recent ReRAM based designs [26, 27], crossbar size of 128×128 is used as our design choice.

4.4.4 Power Analysis of FERA

The power efficiency of FeFET and ReRAM based VMM engines is presented in Figure 4.13. Since this research focuses on the study of a single VMM engine, the power for H-NoC and other shared digital blocks (such as controller) are averaged to each PE. For the baseline ReRAM design, ADC is used in the BL peripherals and insert buffer to drive the WL. With a simple technology replacement from ReRAM to FeFET (still using the same peripherals), the baseline FeFET achieve only 1.2x power reduction as the power consumption on the peripherals (especially the WL buffer) dominated. Then, for

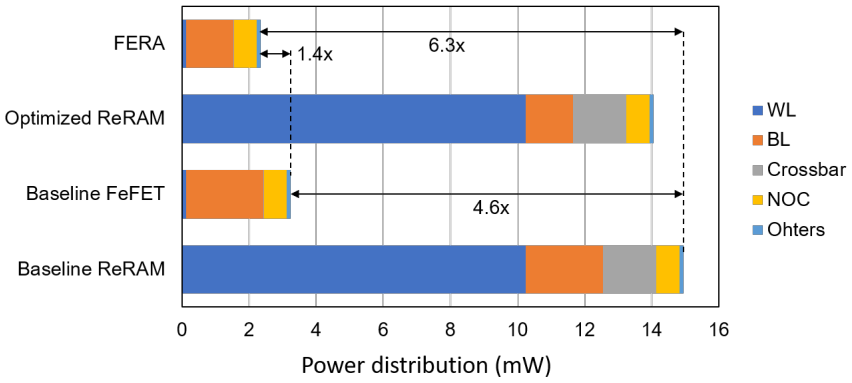


Figure 4.13: Power distribution for baseline ReRAM, baseline FeFET, and FERA.

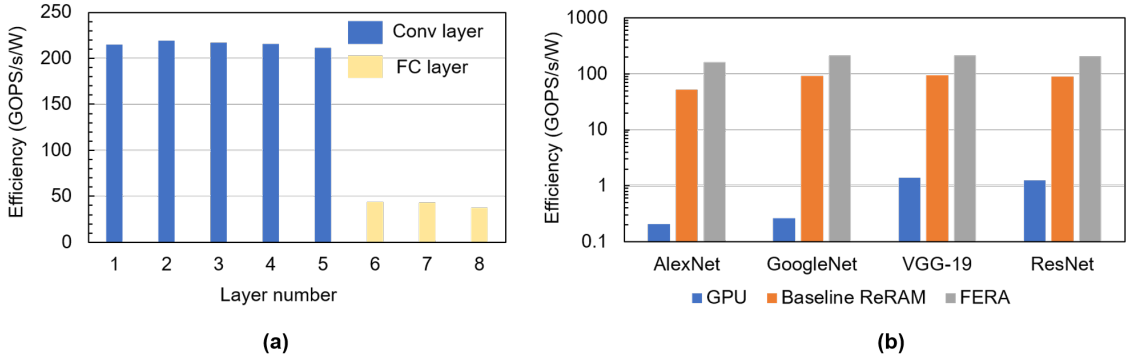


Figure 4.14: (a) Computing efficiency (GOPS/s/W) for the layer-by-layer analysis of AlexNet. (b) Computing efficiency of benchmark DNNs and comparison with GPU/baseline ReRAM

FERA, the power-hungry ADC is replaced with the proposed SA based TDC design and also eliminate the WL buffer, thanks to the capacitive load of FeFET crossbar. Significant power efficiency improvement is observed (another 5.7x). In total, with the cross-cutting solutions combining emerging device technologies and circuit innovations, FERA demonstrates 6.3x power efficiency over the baseline ReRAM design. Later on, it is observed that the device technologies, circuit optimization, together with the micro-architecture innovation, make FERA the most computing efficient solution for DNN accelerator when compared with recent NVM based designs.

4.4.5 Application-driven Performance Analysis

The computing efficiency (GOPs/S/W) of the FERA using the benchmark DNNs is evaluated. Figure 4.14 (a) shows the layer-by-layer efficiency of Alexnet. It is observed that FC layers shows lower efficiency mostly due to large weight matrix requiring partial summations and crossbar re-programming.

Further, FERA is compared with GPU and ReRAM based design, shown in Figure 4.14(b). Thanks to higher efficiency of the FeFET PEs, the average power-efficiency of FERA across the benchmarks are 254x and 9.7x higher over GPU and ReRAM designs, respectively.

4.4.6 Computing accuracy

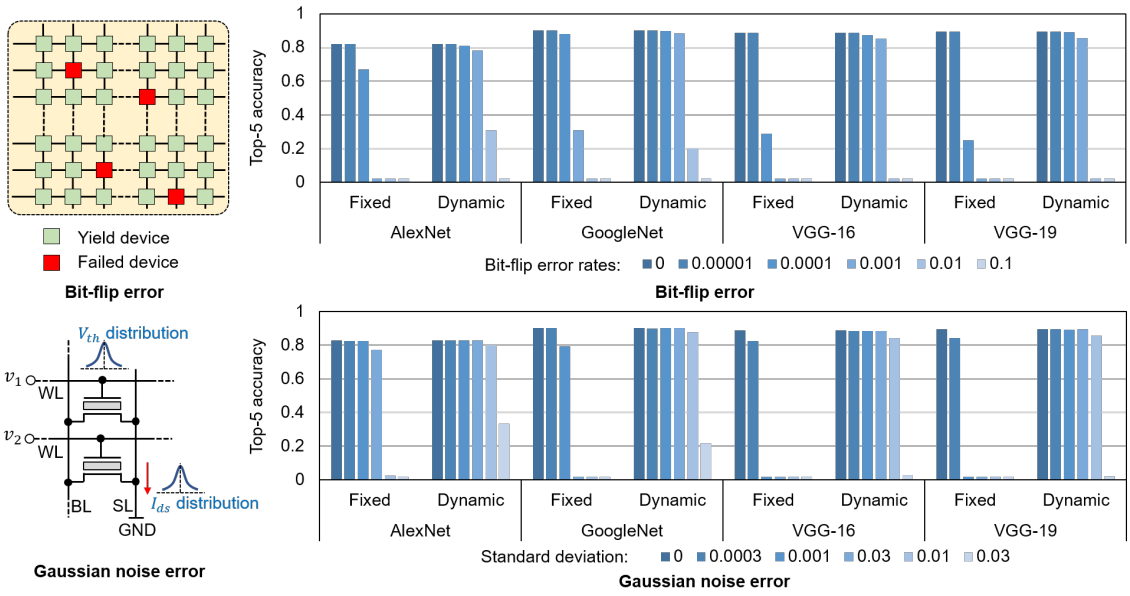


Figure 4.15: The top-5 ImageNet classification accuracy considering device variation (bit-flip error and Gaussian noise) using fixed-point and dynamic fixed-point data representation.

This work also evaluates the impact of device variation on computing accuracy. Two types of device variation are considered, as shown in Figure 4.15: the first is bit-flip error caused by writing failure or device degradation [82] that renders a bit-cell stuck at '1' or

'0'. The second source of device noise is statistical variations represented with a Gaussian noise [108]. The device variation model is calibrated with experimental data in recent published works [82, 86, 95, 109].

Figure 4.15 shows the classification accuracy deterioration under device variation (bit-flip error and Gaussian noise) considering the conventional fixed-point as well as dynamic fixed-point data representation. For bit-flip error, We randomly flip the bit with a possibility ranging from 0 to 0.1 and plot the top-5 accuracy for the benchmark DNNs. Both dynamic fixed-point and fixed-point have good performance when the error rate is lower than 10^{-4}. Moreover, dynamic fixed-point shows a better robustness than fixed-point when the error rate is large. For example, the accuracy drops a lot for GoogleNet when the error rate is higher than 10^{-3} with fixed-point data format, but shows good performance when using dynamic fixed-point. Similar with the case of bit-flip error, the dynamic fixed-point data format also enhance the error tolerance for Gaussian noise. However, we should note that for both data formats, the accuracy drops significantly when the noise level is high (e.g. $\sigma > 0.05$).

4.5 Related works

The high demand for energy efficient execution of deep neural networks have motivated the fast development of DNN accelerators across various platforms including GPU [48, 49], ASIC [10, 16, 17, 12], and FPGA [15].

The more closely related work are emerging non-volatile memory based DNN accelerator architecture that have been intensively explored in recent years [25, 27, 26, 31, 29, 28]. Among all the candidates, ReRAM based design attracts lots of attention, for example, PRIME [25], ISAAC [26], and PipeLayer [27]. PRIME proposes a PIM architecture where the ReRAM crossbar can be configured for both computing and memory [25]. ISAAC presents a full-fledged ReRAM based CNN accelerator with a pipeline architecture and a innovative data encoding algorithm [26]. PipeLayer further optimizes the data flow and

Table 4.4: Performance comparison with other DNN accelerators.

	Technology	Hardware platforms	Training support	Parameter storage	Power (W)	Area (mm2)	Computing efficiency (GOPS/s/W)
ESE [15]	22 nm	FPGA	No	DRAM (off-chip)	41	-	6.88
EIE [16]	45 nm	ASIC	No	SRAM (on-chip)	2.36	63.8	174.1
DaDianNao [10]	28 nm	ASIC	No	eDRAM (on-chip)	20.1	67.7	286.4
Neurocube [17]	15 nm	ASIC	No	DRAM (HMC)	3.4	68.2	38.8
ISAAC [26]	28 nm	ReRAM	No	ReRAM (in-situ)	65.8	85.4	380.7
PipeLayer [27]	-	ReRAM	Yes	ReRAM (in-situ)	-	82.6	142.9
FERA	28 nm	FeFET	No	FeFET (in-situ)	2.27	13.1	443.5

pipeline, adding the feature for on-line training [27]. More recently, a full system stack solution with reconfigurable ReRAM based PIM design is proposed in FPSA [31].

FERA fundamentally differs from the CMOS ASICs approach. Instead of building digital architecture, FERA develops in-memory computation to maximally exploit the fine-grain concurrency in data-intensive applications. FERA also differs from prior ReRAM engines beyond the use of a new technology. First, FERA shows that orders of magnitude increase in the efficiency of VMM crossbar may not lead to similar performance at the system level as peripherals dominate system power. FERA presents lightweight peripherals to increase chip's efficiency. Second, FERA presents a communication fabric, realizing input vector sharing and partial results on-the-fly processing. Finally, FERA shows how data format can be used to improve accuracy under variation.

A detailed comparison between these accelerators implemented with ASIC, FPGA, and NVM technologies is performed. The key design features are summarized in Table 4.4. We note that the ReRAM efficiency's reported in Table 4.4 is higher than the one shown in Figure 4.14. This is because prior works did not consider overheads of WL drivers.

4.6 Summary

In this work, a FeFET based scalable architecture (FERA) is proposed to accelerate DNN computation. With a cross-cutting solution combining emerging device technologies, circuit optimization, and micro-architectural innovations, state-of-the-art performance is achieved. A dedicated NoC design is proposed to provide solution for input data sharing and partial

results summation. Our simulation indicates that, on average, FERA improves the computing efficiency by 254x and 9.7x over GPU and ReRAM designs, respectively. As FeFET continues to mature towards a commercial technology, FERA shows pathway to a scalable architecture that successfully leverages unique properties of the technology to accelerate challenging data-intensive computing applications.

CHAPTER 5

FLEX-PIM: AN ALGORITHM AND HARDWARE CO-DESIGN APPROACH FOR DYNAMIC PRECISION PROCESSING-IN-MEMORY BASED DNN ACCELERATOR ARCHITECTURE

5.1 Introduction

The successful adoption of PIM architecture faces many practical challenges from both the algorithmic aspect of DNNs and the circuit level hardware design, hindering us to fully utilize the high efficiency of in-memory computing.

The first design challenge is based on the observation that not only different DNN models but also each layer of a given model has varying sensitivity against quantization [22]. A recent study [22] reveals that lowering the bitwidth of the first and last layers of ResNet [3] can dramatically degrades the accuracy while the other layers can be safely quantized to 8-bit without sacrificing any accuracy on ImageNet dataset [23]. Handcrafted optimization might be possible for shallow neural network but becomes unfeasible for deeper model. Alternatively, simple uniform quantization works in some degree but can not guarantee an optimal reduction result. **An algorithm that can perform automated layer-wise quantization efficiently without compromising versatility/generality is highly desired**. Consequently, this also presents as a challenge to the hardware implementation, namely, how to design an architecture which can support dynamic bit-precision without compromising the efficiency. Architecture that supports multi-precision has been explored in ASIC domain [110] but is still missing for the PIM based architecture.

The second challenge originates from the fundamental micro-architectural property and circuit design, which unfortunately, has been used to obtain the reported gains in the computing efficiency. In the existing ReRAM based PIM micro-architectures [26, 25], the

high-throughput is achieved by using internally analog computation, in particular, current summation across a bit-line. The analog nature requires analog/digital conversion (ADC and DAC) as data cross the VMM engine (i.e. memory sub-array) interface. Although recent designs use binary operation of memory bit-cells to eliminate DACs, the ADCs remain a critical component. The ADCs lead to power/area overhead and ultimately, limits the computing precision in designs. Consequently, current PIM design typically employs fixed point data representation with a single bit-width (e.g. 8 bit) across the platform. Several recent studies try to replace ADC with spiking based data encoding [27, 31] or time-to-digital (TDC) [30], but fundamentally, the issue is not fully solved. Moreover, the present of analog circuit inevitably introduces customer design (e.g. manually layout ADC), impeding the use of electrical design automation (EAD) which is critical for fast and accurate design space optimization to achieve the best performance. Alternatively, works [28, 111] explore using in-memory logic operation (e.g. NOR) to replace the current summation and ADC/DAC. The intermediate data need to be stored back into the ReRAM crossbar, resulting to frequent ReRAM reprogramming which is very energy hungry. Overall, **the peripheral circuits and micro-architecture need to be redesigned to enable (i) accurate in-memory computing, (ii) dynamic bit-precision support, and (iii) EAD flow for design space exploration.**

This chapter of the research presents a software/hardware co-design approach to achieve a flexible processing-in-memory architecture, referred to as **Flex-PIM**, for accelerating inference and training of deep neural network (DNN). From the algorithm perspective, the innovation is built upon the insight that not only different DNN models but also each layer of a model have varying sensitivity against quantization. A simple uniform quantization across all layers can not guarantee an optimal model reduction. Toward this end, we propose a genetic algorithm (GA) based layer-wise DNN quantization techniques which ensures to find the best quantization strategy without sacrificing the accuracy. From the hardware aspect, the key micro-architectural innovation in Flex-PIM is a memory sub-array based all-digital

VMM engine that eliminates the internal analog operation and hence the analog/digital conversion required in prior works [25, 26, 30, 112] is gone. This is achieved by using a bit-wise AND operation for multiplication using binary memory bit-cells, but replacing the analog current summation with a row-by-row read and accumulation operation. While the design sacrifices the speed of the analog current summation by requiring multiple cycles for accumulation, it recovers the lost throughput by removing the large delays associated with low-power (high-fidelity) ADCs necessary in past VMM designs [25, 26, 27, 30] (see Section 5.5). This simple change in the internal micro-architecture of VMM, together with design space optimization, lead to major gains in the flexibility, energy-efficiency, and accuracy of DNN acceleration.

In essence, the work makes following key contributions to the field of DNN acceleration:

- We propose a Genetic Algorithm (GA) based layer-wise quantization method. Given a set of initial guesses (bitwidth for each layer of a given DNN model), GA can quickly evolve and search for the best quantization strategy. The quantization results can be converted back to the training pipeline to further improve the accuracy. This method features with a fast hyper-parameter (i.e. bitwidth for each layer) space searching for various DNN model and the capability to avoid local optima.

- The proposed Flex-PIM inherits the arithmetic of VMM-in-memory configuration but eliminates ADC/DAC by an all-digital design, ensuring throughput while making the computing accurate and the design fully synthesizable. Dedicated micro-architectural supports are developed to enable fixed-point computation with dynamical bit-precision (for inference) as well as floating point operation (for training) inside memory sub-arrays (VMM engines).

- The work presents a system architecture of Flex-PIM which integrates both memory arrays and fixed function units to ensure Flex-PIM has the flexibility to support differ-

ent types of DNN acceleration. A software-hardware interface and a domain-specific instruction set for Flex-PIM are developed to bridge the gap between application and hardware deployment.

- The work presents an ecosystem for fast design of PIM-based DNN accelerators. Given a desired memory capacity and bit-cell technology (SRAM/FeFET), the developed Electronic Design Automation (EDA) flow explores the design space, selects the configurations of VMM blocks (e.g. number of rows and columns) for maximum efficiency (or performance). The previous PIM designs suffered from the lack of EDA flow as each PIM configurations requires custom design/optimization of ADCs. Moreover, the presented EDA flow can be used to generate Flex-PIM implementations for other bit-cell technologies.

Flex-PIM is demonstrated using 28nm CMOS technology considering SRAM and FeFET based memory technologies. The design space optimization is performed to generate two different implementations of Flex-PIM. One is SRAM based performance-oriented design which supports both training and inference. The other one is FeFET based efficiency-oriented inference-only engine. The cycle-level simulation is used for timing analyses while power is estimated using post-layout analyses of blocks and network. We evaluate Flex-PIM across various DNN models for image classification, activity recognition, and object detection. The SRAM based high-performance configurations shows 56x and 45x better efficiency (GOPs/W) over desktop GPU (Nvidia GTX 1080Ti) and Tensor Processing Unit (TPU-v2), respectively. The low-power Flex-PIM configuration with FeFET shows a 30x performance improvement compared with embedded platform (Nvidia Jetson TX2).

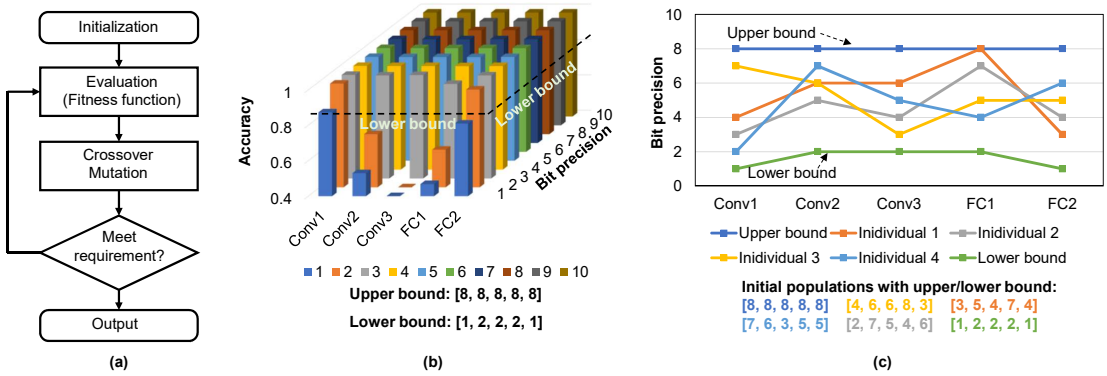


Figure 5.1: (a) A general flow for genetic algorithm. (b) Layer sensitivity towards quantization for LeNet to determine the lower bound. (c) Randomly generated initial populations (i.e. initial candidates) constrained by the upper bound and lower bound.

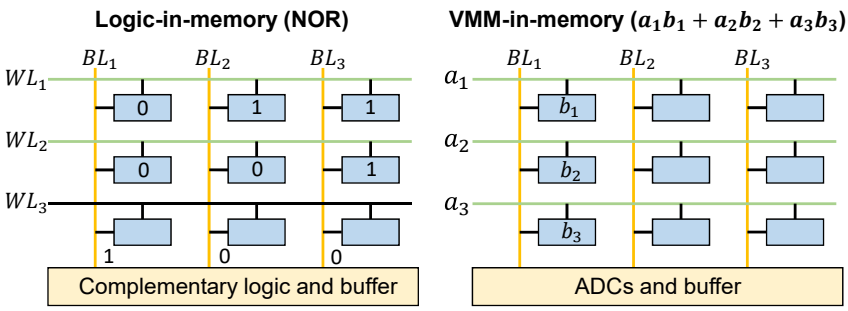
5.2 Background

5.2.1 Genetic algorithm

Genetic algorithm is a heuristic search/optimization algorithm inspired by the process of natural selection and evolution. GA is widely used in optimization and search problems to generate near-optimal solutions in an efficient manner. As shown in Figure 5.1 (a), GA starts from an initialization process where a set of initial populations (i.e. candidate solutions for the targeting problem) are generated. Next an iterative process is used to evolve the initial population to an optimal solution. Given a target problem, a fitness function is used to evaluate the effectiveness of the candidates. At each iteration of the loop, the candidates in the current populations is first evaluated based on the fitness function. Candidates with low fitness value are eliminated while candidates with high fitness value are kept to generate a new generation of candidates via crossover and mutation (i.e. re-generation). This process is repeated until a satisfactory solution is reached.

5.2.2 Processing-in-Memory Configurations

The left side of Figure 5.2 shows how the logic-in-memory PIM configuration works. Assuming we have a ideal memory device which can store either 0 (off state, no current flow



Comparison

	Logic-in-memory	VMM-in-memory
Abstraction level	Basic logic (AND, NOR)	Vector-matrix multiplication
Pros	Accurate, all-digital, reconfigurable	Fast
Cons	Slow	Error-prone, ADC/DAC overhead

Figure 5.2: PIM configurations: logic-in-memory (left) and VMM-in-memory (right). Inserted table shows their pros and cons.

through) or 1 (on state, current can flow through). The bit-lines (BLs) are first pre-charged to 1 and then the top two word-lines (WL_1 and WL_1) are simultaneously activated. The sense amplifer (SA) will sense the BL voltage and store the result in local buffer. The output from SA is 1 only in the case that both two cells connecting to the same BL are 0 (i.e. cutoff), resulting a NOR logic. Since NOR itself is logic complete, other complex logic can then be built on it. Additionally, there are complementary logic units implemented in the BL peripherals for fast bit-serial operations [29].

Alternatively, the memory array can be programmed for vector-matrix-multiplication (i.e. VMM-in-memory), as shown on the right side of Figure 5.2. A binary vector (b_1, b_2, b_3) is programmed to memory cells in the same BL first and another binary vector (a_1, a_2, a_3) is supplied as WL voltage. The current summed at BL results the multiplication-accumulation (MAC) operation. Since the current summation (or charge sharing) happens at analog domain, ADC is required to do the analog-to-digital conversion. In practice, we typically use multiple adjacent cells to store a multi-bit number (one memory cell stores 1-bit) and multiple clock cycles to feed in the input vector.

Generally, VMM-in-memory has better cell utilization and faster than its logic-in-memory competitor. For example, it takes 102 cycles to perform a 8-bit multiplication with logic-in-memory while only 8 cycles when using VMM-in-memory [29]. However, VMM-in-memory suffers from the large power/area overhead introduced from ADC. A recent study indicates that ADC contributes more than 40% of the total system energy and area [26].

In this work, we focus on the VMM-in-memory configuration and propose a new computing scheme to realize an ADC free design. Details will be discussed in section 3.

5.2.3 Candidate Memory Solutions

Various memory techniques have been explored for PIM based designs including DRAM [28], eDRAM [10, 11], SRAM [29], ReRAM [25, 26, 27], and FeFET [30] as summarized in Table 2.2. *DRAM* based design mainly benefits from it's high density and large memory capacity. But as DRAM is difficult to integrate with digital CMOS, a full PIM architecture for DNN acceleration cannot be realized. The *eDRAM* support CMOS integration, but lacks density. Moreover, DRAM/eDRAM need periodic refresh introducing latency and design overhead, The *SRAM* provides faster speed and lower read/write power, but suffers from less density and large leakage power. Moreover, DRAM, eDRAM need periodic refresh and write back after every read (disruptive read issue), introducing latency and design overhead, particularly for DNN operations when weights are stored for a long time during inference.

The *ReRAM* based designs had shown promise for very high energy-efficiency, but suffers from high programming power/latency, and device variation (see chapter 4).

In the previous of this research, FeFET has shown promise as a binary bit-cell for VMM designs [30]. FeFET shows much smaller write energy, and CMOS compatible, but suffers from longer write time.

In summary, as each technology has its merits and the challenges, the choice of bit-cell

technology solution should be application-dependent, but it is desired to have a single core architecture of the PIM-based DNN accelerator.

5.2.4 EDA for PIM Design

The practical adoption of an accelerator often depends on using existing EDA flow or creating a new one for fast design space exploration, optimization, and generation of a full-chip physical design. It is well-known that the performance of a VMM engine depends on its configuration ($size = \#rows \times \#columns$). A larger memory sub-array size increases internal delay of each VMM engines, but can be beneficial in terms of computing parallelism. Hence, given a total memory capacity (as well as the memory type), an EDA methodology is necessary to generate the optimal configurations of VMM engines, and integrate them wisely to enhance system performance (or reduce power). Although, commercial memory compilers can perform the preceding for SRAM designs, there has been limited progress in developing an EDA tool for PIM (even with SRAM). A major barrier here is the internally analog design of the existing VMM engines, where summation is performed by adding currents and then digitizing the summed current using an ADC. As the number of rows the VMM engine changes, the dynamic range of the current sum changes as well. This essentially requires a custom re-design of the ADC to ensure optimal noise, performance, and power trade-off of the ADC. In other words, each configuration of the VMM engine with require a different ADC design which significantly increases the design time/cost and pose a significant challenge to develop a EDA flow for design space exploration of existing VMM-in-memory based PIM designs. The problem is further exacerbated if we consider using different memory cell technologies which further changes the characteristics of the input signal. It should be noted that, automated synthesis of ADC design/layout is still an open research topic in analog design community. In this paper, the use of an all-digital VMM engine, solves this core problem, making an EDA flow for Flex-PIM feasible. Section 5.7 discusses the developed EDA flow for Flex-PIM.

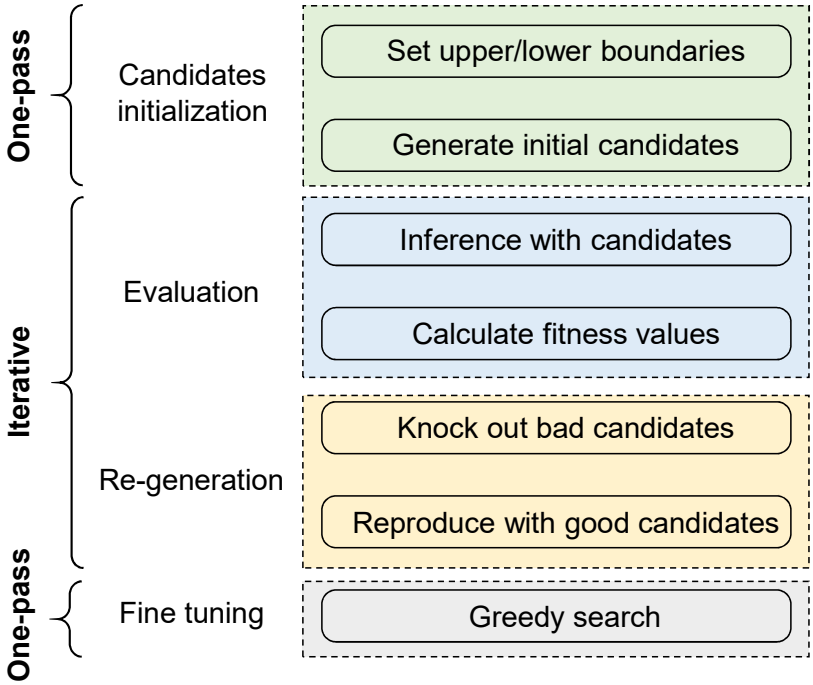


Figure 5.3: GAQ flow.

5.3 GA based layer-wise quantization (GAQ)

5.3.1 Proposed Approach

For simplicity, we use LeNet [113], a 5 layer CNN containing 3 conv layers and 2 FC layer, for MNIST dataset [**mnist**] classification as an example to illustrate the proposed algorithm.

Figure 5.3 shows the overall flow for GAQ which largely follows the general genetic algorithm except the last step fine tuning. One should note that the evaluation and re-generation for new candidates are iterative processes which are terminated only when a satisfactory quantization is achieved or the evolution saturated. We next discuss the key steps of the flow in detail.

Candidates initialization. Since the objective is to find an optimal strategy for layer-wise quantization, each candidate in GA is a list containing N integers where N equals to the number of layers in the target DNN model and each number in that list represents

the bitwidth of a corresponding layer. For example, with the 5 layers LeNet, a possible candidate could be [8 8 8 5 5], indicating that 3 Conv layers are quantized to 8-bit and 2 FC layers are quantized to 5-bit. To ensure the convergence speed, we first constrain the search space, namely, define the upper bound (UB) and lower bound (LB) for the GA optimization. The upper bound is defined in a heuristic way, for example, upper bound is defined as [8 8 8 8 8] for LeNet on MNIST dataset since 8-bit is good enough to guarantee the accuracy. The lower bound, on the other hand, is determined by the sensitivity of each layer. As shown in Figure 5.1 (b), we gradually reduce the bitwidth of one layer until the inference accuracy drops to a pre-defined threshold (e.g. 2% accuracy drop) meanwhile keep all other layer in floating point precision. For layer which is sensitive to quantization, high bit precision is used as the lower bound for that layer; for layers showing good robustness, low bit precision is used instead. For LeNet, the lower bound is set to be [1 2 2 2 1] under a 2% accuracy drop threshold. One should note that in this process we directly quantize the pre-trained model to perform evaluation, no training is involved. With the upper/lower bound, an initial candidate can be generated with the equation 5.1.

$$C = [rand(high = UB_i, low = LB_i) \ \ for \ \ i \ \ in \ \ [0...N]] \tag{5.1}$$

$rand$ is the function to generate random integer between $high$ and low. N is the number of layers in the given DNN model ($N = 5$ for LeNet). Figure 5.1 (c) shows several possible initial candidates generated based on the upper and lower bound.

Evaluation. In the proposed algorithm, the fitness function (F) evaluates two metrics of each candidate (C): the compression ratio and the computing accuracy. Higher compression rate as well as better inference accuracy indicate the candidate has a higher fitness value, in another word, a better quantization solution. The fitness function is defined in

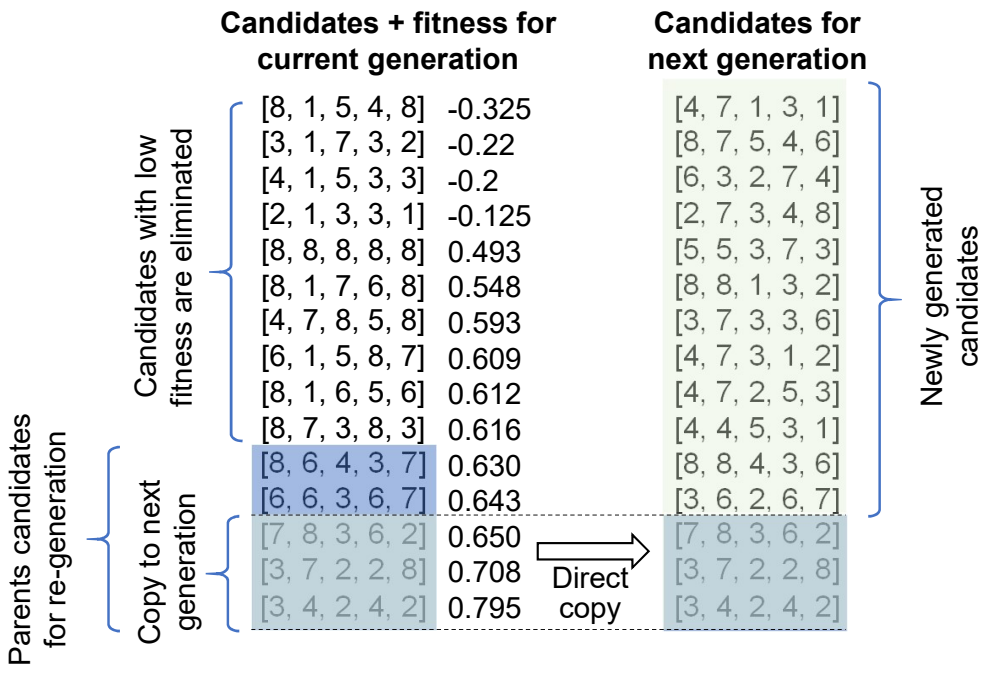


Figure 5.4: Procedure for re-generation. Candidates are ranked based on their fitness value. The bottom 5 candidates are used as parents to reproduce new generation and the bottom 3 candidates are directly copied to the next generation. The rest are removed from the evolution process.

equation 5.2.

$$F(C) = -\alpha \cdot \sum_{i=0}^{N} C_i W_i - \beta \cdot Err$$
$$where \;\; Err = \begin{cases} Err & acc \geq threshold \\ inf & acc < threshold \end{cases} \tag{5.2}$$

where C is the candidate solution (i.e. a list with N integers); C_i is the bitwidth of i-th layer of the DNN and W_i is the number of weight parameters in that layer; Err is the error introduced by quantization and a penalty (inf or ∞) is applied to the fitness function if the error exceeds a pre-defined threshold; α and β are weighting factors for compression rate and accuracy, respectively. Again, the error rate is computed from the inference stage and training is not required.

Re-generation (Crossover and mutation). Based on the fitness value, good candidates (i.e. candidates with high fitness) are reserved to reproduce the next generation populations

and the bad candidates are eliminated. Each generation is composed of 15 independent candidates and they are ranked based on their fitness value, as shown in Figure 5.4. The bottom 5 candidates (candidates with best fitness) are used for re-generation. To generate a new child candidate, two parent candidates are randomly picked from the reserved parent candidates pool (5 good candidates in Figure 5.4). Then the child candidate is generated based on the range described by the two parent candidates, regulated by equation 5.3.

$$\begin{aligned} C = [randi(high = max_i, low = min_i)\ for\ \ i\ \ in\ [0...N]] \\ where\ \ max_i = max(A_i, B_i) + 1 \\ min_i = min(A_i, B_i) - 1 \end{aligned} \tag{5.3}$$

where A_i and B_i are the bitwidth of the i-th layer defined by the parent candidate solutions A and B, respectively. Essentially, the two parents candidates define the upper/lower bound to reproduce a new candidate where the upper and lower bound for i th layer is set by the larger and smaller value from A_i and B_i, respectively. One should note that we expand the search space by adding 1 to max_i and reducing 1 from min_i. This operation ensures that outliers are generated during evolution, reducing the possibility of trapping into local optimal. To ensure the best results and fast convergence, at each iteration, 3 candidates with highest fitness value from last generation is append to the newly generated individuals to form the new generation.

Fine tuning after GA. While standalone GA works perfectly on shallow models such as LeNet (5 layers) and AlexNet (8 layers) [1], we observe that the output (layer-wise quantization strategy) from GA for deep models still has room to be further compressed with only insignificant accuracy drop. Therefore, we apply the greedy search (GS) algorithm to the output of GA to further reduce the bitwidth of layers that are robust toward precision reduction. GS is an iterative method but performs search in a 'greedy' approach. At each step, GS iteratively reduces 1-bit for a layer and checks which layer shows the best robustness towards bit reduction; Then the precision for the most robust layer (i.e. layer with

minimum accuracy drop due to the 1-bit precision reduction) will be reduced accordingly. With GS, another 5% - 10% reduction is achieved without accuracy loss for models like VGG [2] and ResNet [3].

5.3.2 Quantization for activation

We use a heuristic approach for quantization of activation. As activations are more sensitive to quantization than weight parameters, we employ equation 5.4 to determine the precision for the activation.

$$P^i_{act} = min(P_u, P^i_{weg} * 2) \tag{5.4}$$

P^i_{act} and P^i_{weg} are the precision for activation and weight in ith layer, respectively. P_u is a dataset-dependent value which is used to constrain the maximum possible precision for activation. For MNIST and CIFAR-10, P_u is 8-bit; for ImageNet, P_u is 12-bit.

5.3.3 Accuracy and Runtime Analysis of GAQ

We carefully evaluated the evolution of GA based layer-wise quantization for LeNet and MNIST dataset, as shown in Figure 5.5. We also evaluate two baseline method, random search (RS) and greedy search (i.e. GS). RS is a brute force solution which randomly generates a possible solution at each iteration. If the solution yields a higher fitness value (using the same equation defined in 5.2) than the record, then it is set as the new record. GS is the same algorithm we used for fine tuning. But here GS is applied on non-quantized DNN model (e.g. starts from 32-bit floating-point for every layers) directly rather than the output of GA. Also, in Figure 5.5, we show the compression rate for uniform quantization (red horizontal dash line) and the optimal layer-wise quantization achieved by manual tuning (green horizontal dash line), respectively.

We observe that RS is obviously not feasible even when the neural network model is simple. GS, however, shows decent compression rate even though the convergence is

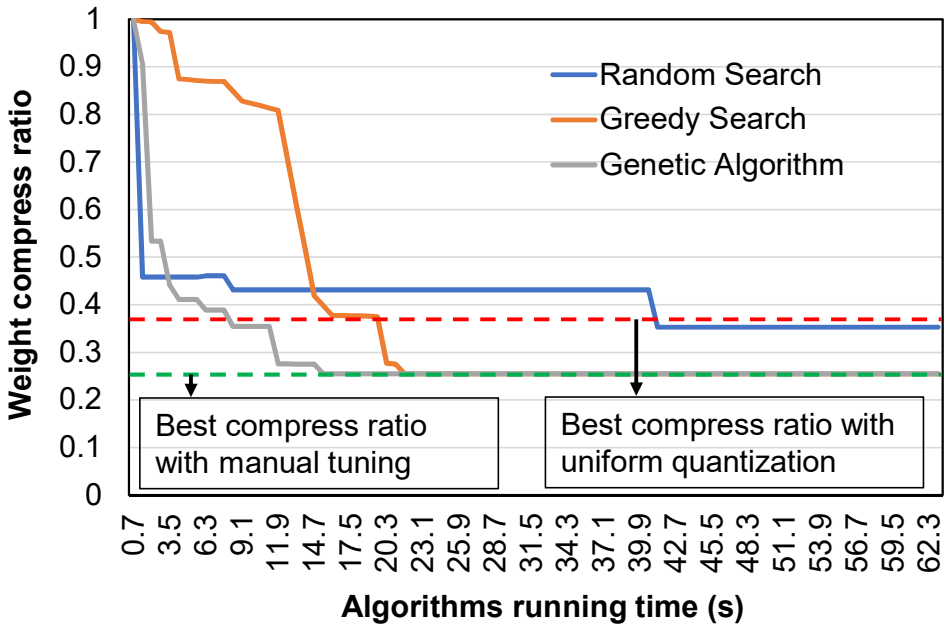


Figure 5.5: Compression ratio for different algorithms. The red dash line is the compression ratio for the uniform quantization (3-bit). The green dash is the optimal compression strategy, [1 3 2 2 2] bits for the 5 layers of LeNet (handcrafted).

slower than GA based algorithm. However, GS based approach becomes less applicable when testing on deeper models as the computational complexity for GS increase dramatically on large DNN models ($\mathcal{O}(N^2M)$ where N is the number of layers in DNN model and M is the number of computations in that model). For instance, with VGG-19 [2], GS take 4x longer time than GA to reach convergence while the compressed model is 15% larger. Overall, compared with the baseline algorithms, GAQ shows better compression rate and faster convergence ($\mathcal{O}(M)$).

5.4 Experimental Results for GAQ

We implement GAQ with Tensorflow [76] using its high-level API Tensorpack [114] and Tensorflow-slim [115]. The benchmarks includes several well-know DNN models: LeNet [113] for MNIST dataset, AlexNet/VGG [1, 2] for CIFAR-10 dataset, and ResNet-18/34/50 [3] for ImageNet dataset. We compute the compression rate in terms of weight parameter size reduction and computational complexity reduction over the single precision (32-bit)

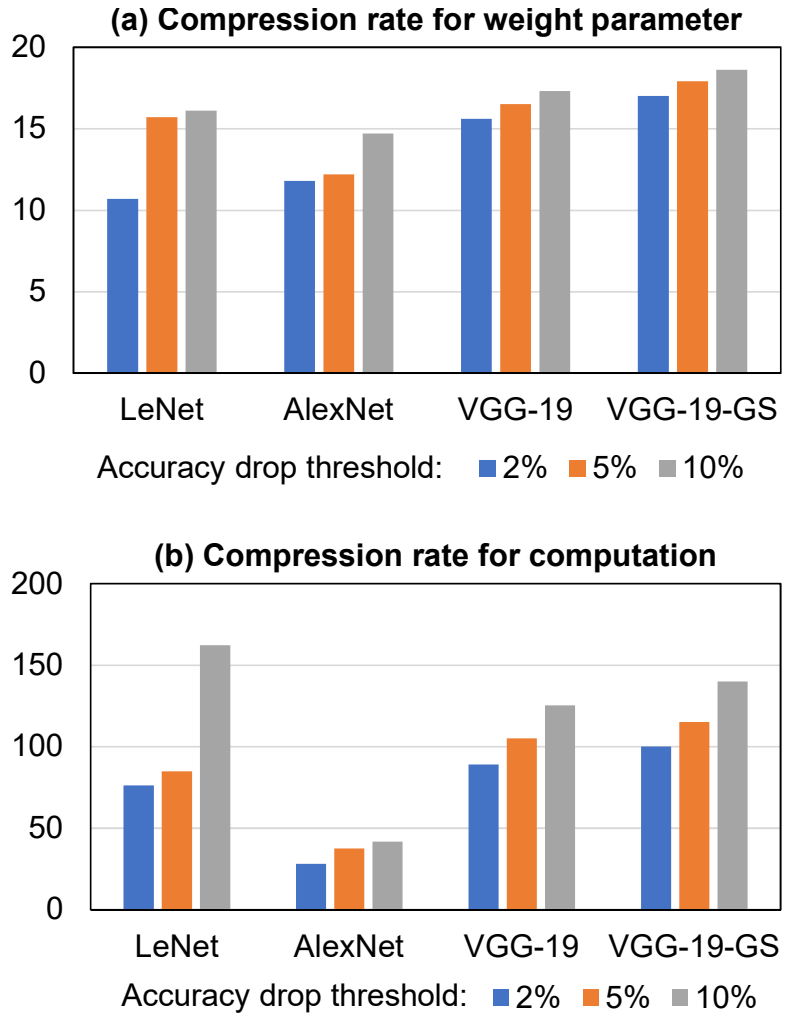


Figure 5.6: Compression rate for LeNet on MNIST, AlexNet, VGG-19 on CIFAR-10 (VGG-19-GS is the results after greedy search). (a) Weight compression. (b) Computational complexity compression.

floating point scenario. The computational complexity reduction is calculated as $(32/P_{act})\times(32/P_{weg})$. For example, a layer is quantized to 16-bit activation and 8-bit weight, we project that GAQ reduces computational complexity by $(32/16) \times (32/8) = 8$ times over the floating-point.

5.4.1 Evaluation on MNIST and CIFAR-10

Figure 5.6 (a) shows the weight compression rate for LeNet, AlexNet and VGG-19 under different accuracy drop threshold (i.e. $threshold$ in equation 5.2). For VGG-19, greedy

search based fine tuning is applied to bring another $\sim 10\%$ reduction. With relaxed accuracy (higher accuracy drop threshold), GAQ yields better compression rate. For example, with $threshold = 2\%$, the average compression is 13.7x across the DNN models and the number becomes 16.7x with $threshold = 10\%$.

With the layer-wise quantization strategy, the activation is quantized according to equation 5.4. The computational compression rate is in Figure 5.6 (b). With $threshold = 2\%$, the average compression for computation is 73.3x. AlexNet shows lower computation reduction. This is because the second Conv layer of AlexNet dominants the computation (43% of total GOPs) and this layer is sensitive to quantization (only quantized to 6-bit).

5.4.2 Evaluation on ResNet with ImageNet Dataset

We evaluate GAQ with ImageNet dataset using ResNet-18/34/50. Compared with AlexNet/VGG-19 on CIFAR, ResNet are much sensitive to quantization. With standalone GA (no GS based fine tuning), the average weight compression is 4.9x with 2% accuracy drop threshold (Figure 5.7 (a)). The model size can be further reduced without accuracy drop using GS (named as ResNet-xx-GS in Figure 5.7), resulting to 6.3x reduction eventually.

The computational complexity compression rate is shown in Figure 5.7 (b). With greedy search for fine tuning, GAQ achieves 18.5x computation reduction on average for ResNet on ImageNet dataset.

Figure 5.8 shows the GAQ layer-wise quantization for ResNet-18 with different accuracy drop threshold. With relaxed accuracy requirement, more layers can be reduced to lower bit precision. More details for other DNN models can refer to appendix.

5.4.3 Comparison with other works

Table 5.1 shows a detailed comparison between GAQ and other state-of-the-art DNN quantization algorithms on ResNet and ImageNet, including LQ-NETs [**lq-net**], UNIQ [116], DoReFa [114], Apprentic [22], PACT [117], and AQN [**adaptive˙quantization**]. We only

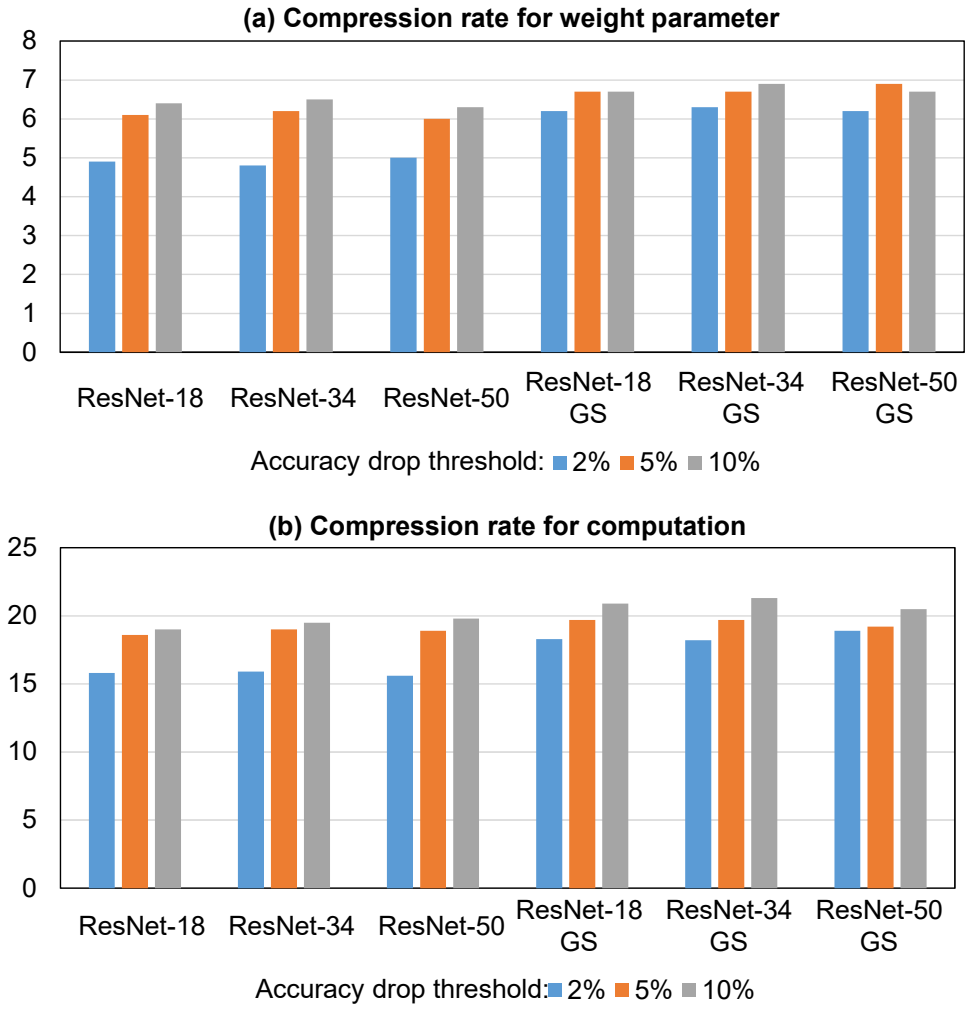


Figure 5.7: Compression rate for ResNet-18/34/50 on ImageNet (ResNet-xx-GS is the results after greedy search based fine tuning). (a) Weight parameters compression. (b) Computational complexity compression.

include works that contain results for ResNet on ImageNet dataset into our comparison. Except AQN, all other works rely on multi/one pass training. For Apprentice and PACT which achieve 2-bit weight precision (i.e.best weight compression rate), handcraft tuning is required to determine which layers are reserved in floating-point precision. AQN, on the other hand, is a fully automated approach with no requirement for training, but the compression rate is less satisfactory.

On average, GAQ can reduce the weight parameter to $\sim$ 5-bit with insignificant accuracy drop. While GAQ shows less compression than the 2-bit quantization algorithms

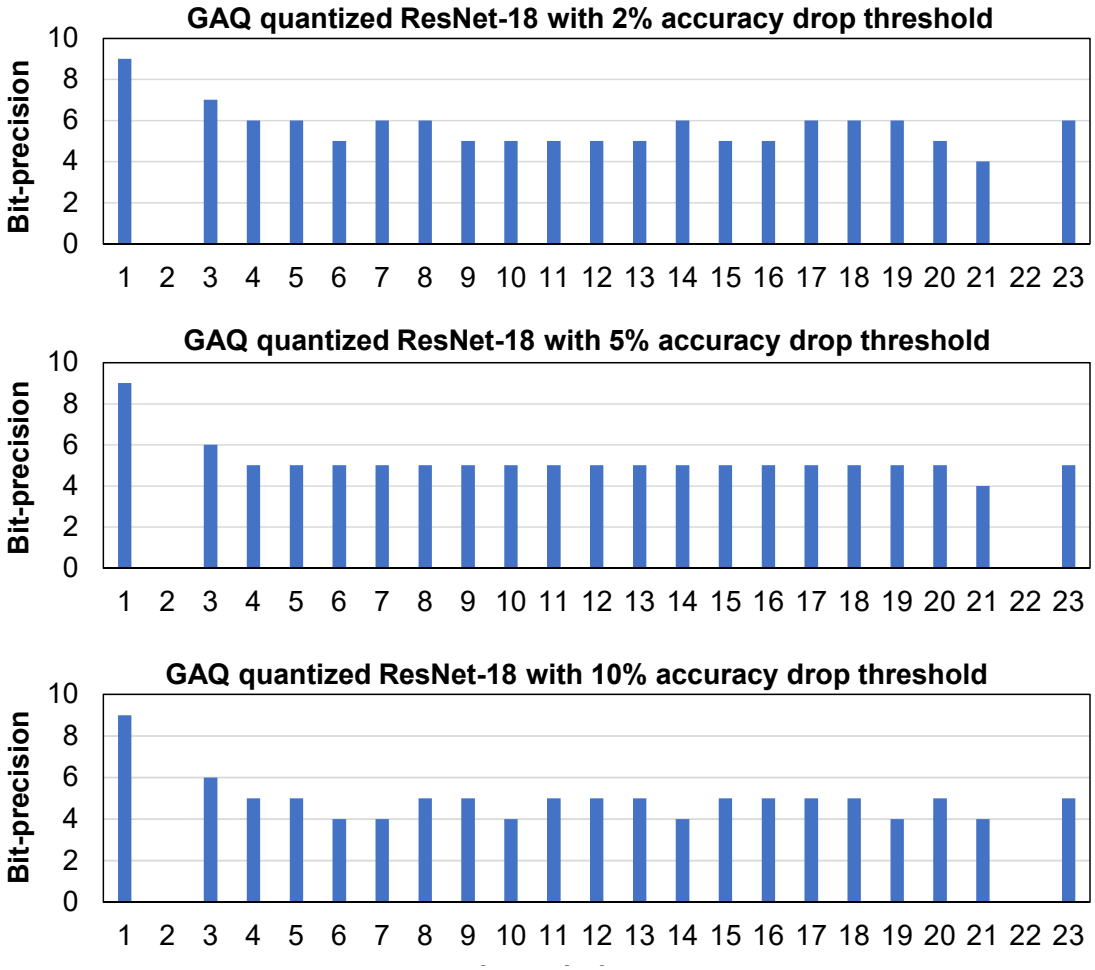


Figure 5.8: Layer-wise quantization for ResNet under different accuracy drop threshold. Layer 2 is pooling layer; layer 22 is gap layer.

such as Apprentice [22] and PACT [117], we argue the key advantage of GAQ over other algorithms is that GAQ works without time consuming training nor large training dataset. Further, GAQ is much flexible with no requirement of manual tuning for layer-wise quantization. Compared with AQN, which is also an inference-only algorithm, GAQ show higher compression rate (11% higher than AQN for ResNet-50).

Table 5.1: Comparison against other SOTA DNN quantization works with focuses on ResNet and ImageNet dataset.

ResNet-18:	**Algorithms**	**Quantization**	**Training complexity**	**Top-1 baseline**	**Top-1 quantized**	**Degradation**
	LQ-NETs	W4/A32	Multi-pass	70.3	70.0	0.3
	PACT	W2/A2 and floating point	One-pass	70.4	67.0	3.4
	DoReFa	Binary	One-pass	70.2	62.6	7.6
	Our work	**Layer-wise Avg. 5.1-bit**	**No need**	**70.4**	**68.9**	**1.8**

ResNet-34:	**Algorithms**	**Quantization**	**Training complexity**	**Top-1 baseline**	**Top-1 quantized**	**Degradation**
	Apprentice	W2/A8 and floating point	Multi-pass	73.6	71.5	2.1
	Our work	**Layer-wise Avg. 5.1-bit**	**No need**	**73.8**	**71.9**	**1.9**

ResNet-50:	**Algorithms**	**Quantization**	**Training complexity**	**Top-1 baseline**	**Top-1 quantized**	**Degradation**
	UNIQ	W4/A8	One-pass	76.0	73.4	2.6
	PACT	W2/A2 and floating point	One-pass	76.9	74.2	2.7
	AQN*	Layer-wise Avg. ~ 6-bit	No need	76.4	74.8	1.2
	Our work	**Layer-wise Avg. 5.2-bit**	**No need**	**76.4**	**74.6**	**1.8**

*Results for AQN is projected from the figures in (Zhou et al. 2018)

5.5 VMM engine μ-architecture

5.5.1 Design Objectives

The first design objective is to achieve *an all-digital design*. Most recent works employ binary based computing schemes where the memory cell stores 1-bit and the WL voltage is also binary [27, 30]. Therefore, DAC (in WL peripherals) is naturally eliminated. On the other side, efforts have been made to replace ADC (in BL peripherals) using spiking based [27], or pre-charge/dis-charge based sensing scheme [30]. However, complex and power-hungry analog circuitry still exist.

The second design objective is to realize *flexible bit precision*. While 8-bit or even lower data precision has been proved to achieve good accuracy in some image classification applications, we argue that it is not always true for other DNN models. Our experiments show

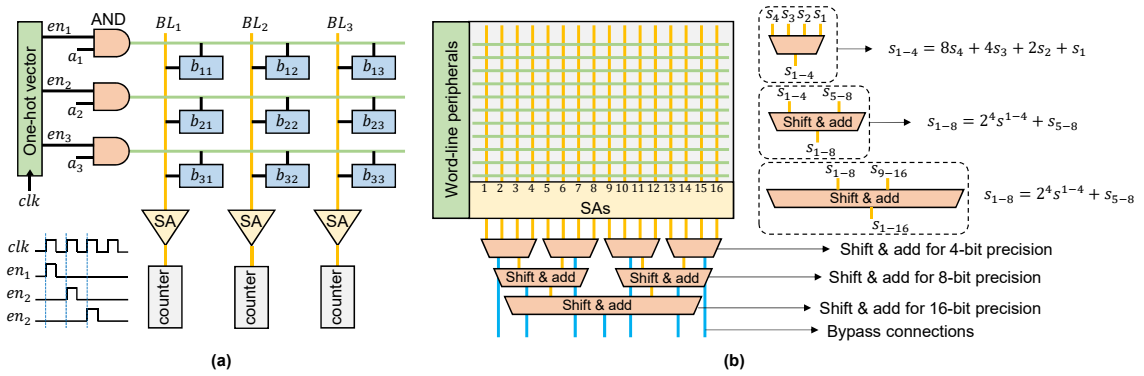


Figure 5.9: (a) WL-wise reading scheme to eliminate ADC. (b) Using hierarchical *shift & add* to support flexible bit-precision.

that, after 8-bit quantization and re-training, the mobilenet-SSD [118, 119] (a fast and accurate object detection DNN model) accuracy drops around 10% in terms of mean average precision (mAP). Further, we also observe that only quantizing the front-end feature extractor while keeping the back-end detection network in 32-bit precision (i.e. dynamical bit-precision) gives much better accuracy ($< 2\%$ drop of mAP). Therefore, it is highly desired to have a system which can seamlessly run different bit precision without introducing overhead.

Last but not least, we want to enable *floating point operation* within memory, which is very critical for DNN training. Even though fixed point (32-bit at least) based DNN training show very little accuracy drop for some DNN models [120], it is still a wise option to train the network with floating point number to guarantee the best accuracy. Most existing PIM based designs focus on inference only. Some designs support training but perform floating point operation externally using CMOS logic [18, 24] or limit the discussion scope within the training pipeline/parallelism without implementing floating point operations.

5.5.2 All-digital Design

The ADC is inevitable if multiple WLs are simultaneously activated. Therefore, we take a step back and activate single WL per clock cycle. As shown in Figure 5.9 (a), The first part of WL peripheral is a clock-driven one-hot vector unit. At the first clock cycle, the

enable signal for the top WL (en_1) is turned on. One should note that en_1 goes to low at the second half of the cycle for the purpose of pre-charging the BL for next read cycle. Then, at the second clock cycle, en_2 is enabled, and so on. Besides the clock signal, there is another control signal (not shown in the figure) to perform early-termination when part of the memory sub-array is unused. The second part of WL peripheral is a logic gate which performs AND operation between the enable signal (en_i) and input vector (a_i). Only when both en_i and a_i are high, the value stored in corresponding memory cells (b_i) are sensed out. At BL peripheral, sense amplifier (SA) is employed to sense out the value (either 0 or 1) and send it to the counter at each clock cycle. Essentially, the MAC operation is performed with N cycles where N is the number of rows in the memory array.

In terms of speed, our design is faster than ADC based approach. This is due to the fact that for prior ADC based design [26, 30], even though the MAC operation is done in one cycle, the speed of ADC (or multi-level sense amplifer) is the major throughput bottleneck. For example, in [26], a 1.3 giga-samples-per-second (GSps) SAR ADC is employed and shared by a memory crossbar. It takes 100ns to converted the analog values from a 128×128 memory array. In our design, it takes 128 clock cycles to perform the same computing. One should note that the clock frequency (clk_{mem}) depends on the memory techniques, array size, and SA design. For a 128×128 SRAM array (i.e. a standard 8KB SRAM array), the internal clock frequency can go up to 5GHz in 28nm technology [29, 112], resulting 25.6 ns to perform the MAC operation, 4x faster than ADC based solution in [26]. Further, our approach outperforms ADC based design in terms of power consumption. The synthesized result with 28nm TSMC technology indicates that 128 7-bit counters is roughly 2.7x energy efficient that one 7-bit ADC [26]. A detailed power analyses will be conducted in section 6.

Besides the speed advantage, our design can achieve a much more accurate computing scheme, eliminating the error introduced from ADC circuits. Moreover, our design can largely reduce the error caused by the device variation for some emerging memory such as

ReRAM and FeFET. By ensuring a large on/off ratio (i.e. large sensing margin), the device fluctuation can be less concerned.

It should be noted that a similar WL-wise reading scheme is presented in a recent work [112]. But they still applied ADC/DAC to the memory peripherals and only focus on the sub-array level design, lacking support for large scale DNN models.

5.5.3 Flexible Bit Precision

The flexible bit precision is achieved by a set of hierarchical organized *shift & add* units, as shown in Figure 5.9 (b). The first level *shift & add* accepts results from 4 BLs, shifts the value and adds them together. If the data precision is 4-bit, then the result can be directly sent to the nearest router (NoC and rounter will be discussed in next section) via the bypass connections (blue line in Figure 5.9 (b)). Alternatively, if the data precision is not 4-bit, the results from two adjacent 4-bit *shift & add* are routed to the second level *shift & add*. It can either send output via bypass lines or to the third level *shift & add*. Eventually, with the hierarchical *shift & add* based design, we can perform fixed point MAC operation with different bit precision in the same place. In our design, we support 4-bit, 8-bit, 16-bit, and 32-bit fixed point operation.

Compared with ASIC based design which either use floating point unit (FPU) for fixed point multiplication or have separated computing resources for different type of operations, our PIM design can support flexible bit-precision as well as floating point operation (next subsection) in the same place seamlessly.

5.5.4 Support for Floating Point Operation

Our PIM design supports floating point operation which is compliant with IEEE 754 floating point standard [121]. The single precision format of IEEE 754 has 32-bit with 1-bit for sign, 8-bit for exponent, and 23-bit for mantissa, as shown in Figure 5.10 (a). But to clearly illustrate our approach, we use a very simplified version of floating point number as an

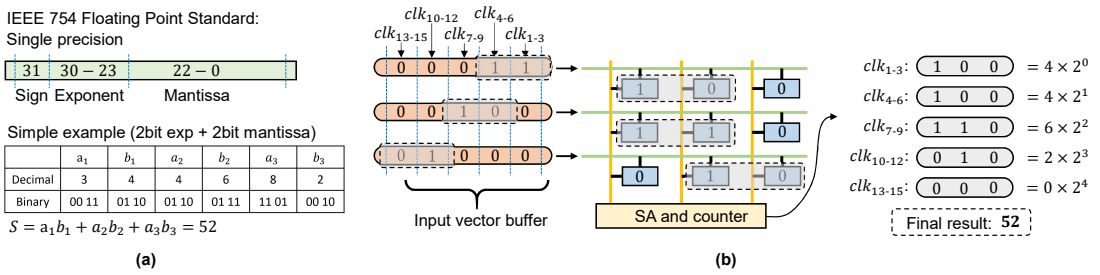


Figure 5.10: (a) IEEE floating point standard and simplified version of floating point data representation. (b) Mapping and computing of floating point MAC inside memory.

example which has the first 2-bit as exponent and the rest 2-bit for mantissa (mantissa only has integer, no fraction bit). We also assume all the numbers are positive and therefore there is no need for sign bit. As a example, 1111 in decimal is $(1\times2^1+1\times2^0)\times2^{1\times2^1+1\times2^0} = 24$.

The embedded table in Figure 5.10 (a) shows the MAC operation between two vectors $\langle a_1, a_2, a_3\rangle$ and $\langle b_1, b_2, b_3\rangle$. The final result equals to 52. The first step is to map the $\langle b_i\rangle$ vector to the memory crossbar. We check the maximum exponent of the three numbers in $\langle b_i\rangle$, which is 01. Then, value with maximum exponent (b_1, b_2) is mapped from the leftmost BL (i.e. MSB); for value with smaller exponent (b_3), mapping will be performed after right shifting. Similar with mapping vector $\langle b_i\rangle$ to the crossbar, we implement a input vector buffer to temporarily store vector $\langle a_i\rangle$. Following the same rules, for element with the maximum exponent (i.e. a_3), the mantissa is stored from the leftmost column of the input vector buffer. For element with smaller exponent (i.e. a_1 and a_2), we perform right shift and append 0 to the left end. Figure 5.10 (b) shows how we map $\langle a_i\rangle$ and $\langle b_i\rangle$ to the input vector buffer and memory array, respectively.

After mapping is done, computation process is similar as fixed point operations. At the first clock cycle, we fetch one value from the input vector buffer (start from the top-right corner) and activate the first WL accordingly. Since we only have three WLs, it takes 3 cycles to get the first part of the partial sum. This process is repeated until all the values are fetched from the input vector buffer. After proper shifting, these partial results are summed together to get the final result, which is 52.

For single precision floating point number, the range of exponent is -127 to 128, indicating that the length of the input vector buffer and number of columns in memory array to represent one number should be at least 255 bits, to accommodate all possible values. This is apparently not practical and will significantly slow down the computing speed. In our implementation, we set the length of input vector buffer (also the number of cells to store one mantissa) to be 32-bit. We argue it can provide the same precision as CMOS logic based floating point unit (FPU) design, where rounding errors also happen when the difference of the exponents of two operands are larger than 23-bit. For example, with single precision notation, $1.5 \times 2^{-1} + 1.5 \times 2^{-32}$ still equals to 1.5×2^{-1} for a standard FPU.

Admittedly, enabling the floating point operation introduces some design and speed overhead due to the additional input vector buffer, complex control signal, data mapping algorithm, and the logic unit to search for maximum exponent. Therefore, for inference-only implementation, these additional designs can be removed for better energy efficiency.

5.6 Flex-PIM System Design

5.6.1 Flex-PIM Instruction Set

A key step to enhance the system flexibility is to design a dedicated domain-specific instruction set. Flex-PIM instruction set is designed following the methodology of Cambricon (a ISA for ASIC based DNN accelerator) [122]. However, different from Cambricon's approach, our instruction set is based on high-level (i.e. DNN layers) abstraction, making the code more readable and informative. Further, we argue that our approach doesn't lose general applicability since, internally, the computing is performed with the granularity of **matrix/vector operation**.

As shown in Table 5.2, there are three types of instructions, **control**, **layers**, and **parameter specification**. Instructions are 64-bit with the first 6-bit as Opcode. The first 2-bit of Opcode specify the instruction types (00 for control, 01/10 for layers and 11 for parameter specification). Control instruction is used to define computing precision, set running

Table 5.2: Flex-PIM instruction set.

Instruction type		Operands	Execution flow
Control		Precision, Train/inference mode, Batch size, registers	Reset control variables and regs
Layers	Conv layer	Output feature map, Input feature map, Convolution kernels	Load data ↓ dispatch data ↓ computing ↓ collect result ↓ store data
	FC layer	Output vector, Input vector, Weight matrix	
	Pooling layer	Output matrix, Input matrix	
	BN layer	Output matrix, Input matrix	
	Activation	Output vector, Input vector	
Parameter specification		Input size, Kernel size, Output size	Determine load/restore pattern

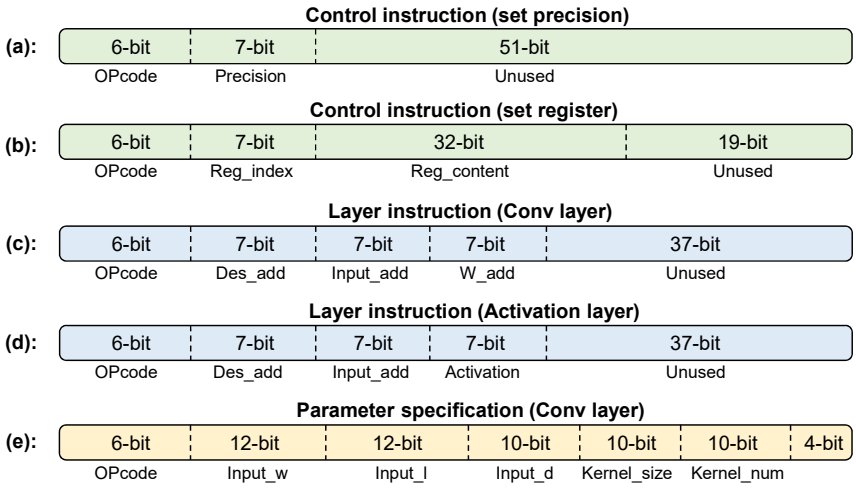


Figure 5.11: Example instructions decomposition.

mode (inference or training) and write address register. Flex-PIM contains 128 32-bit address registers which store the location for weight parameters (also activation if in training mode) for each layer of DNN. A tag bit is appended to each entry of address register to indicate whether the data is on-chip or resides in the off-chip DRAM. Figure 5.11 (a) and (b) show the control instruction to set precision, and write address register, respectively. The second type of instruction defines the layer and where the weight and activation should be fetched from. As the examples shown in Figure 5.11 (c) and (d), for layers with weight parameters (such as Conv layer and FC layer), there are three operands (each has 7-bit,

corresponding to the 128 address registers) for computing result address (*Des_add*), activation address (*Input_add*), and weights address (*W_add*). For layers without weights (such as ReLU or sigmoid), the third operand is to indicate the computing type (*Activation*).

However, we are still facing an issue that 64-bit instructions can not include all the information for a layer definition, such as input feature map depth, convolution kernel size, etc. This leads to the implementation of the third type of instruction, namely, parameter specification. After every layer instruction, a parameter specification instruction is inserted to provide complementary information for the layer definition. During execution, once an instruction is decoded and identified as layer-type instruction, the processor immediately fetches another instruction from the instruction buffer. After getting all the necessary knowledge, the processor then asks the DMAC (direct memory access controller) to fetch data from right location and perform data partition and dispatching. Figure 5.11 (e) shows an example of a parameter specification instruction which provides definitions for a Conv layer.

5.6.2 Chip-scale architecture

Figure 5.12 shows the system architecture for Flex-PIM. Memory sub-arrays are placed in a 2-D plate, interconnected with a hierarchical network-on-chip (H-NoC) [30]. An accumulator is implemented inside the router to realize on-the-fly partial results summation (in the case that single matrix operation is mapped to multiple memory sub-arrays). Experiments indicate that such NoC design can significantly improve the data transmission efficiency (both input data dispatching and result collection) [30]. Additional to the original design, in this work we design the H-NoC to support flexible bit-precisions and our implementation is fully parameterized. To be more specific, with different parameters, our H-NoC can be reconfigured to achieve different levels of hierarchy and data bandwidth to fit various application scenarios. In section 6, we show that with different throughput requirements and power budgets, we can either have deeper hierarchy (7 levels to realize

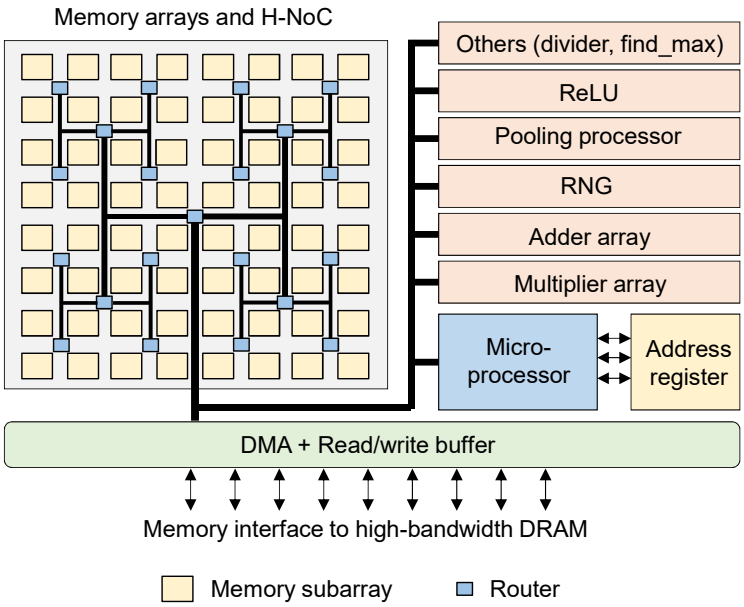


Figure 5.12: System architecture for Flex-PIM.

16MB on-chip memory capacity) and higher bandwidth for performance-oriented design, or shallower hierarchy (5 levels to realize 2MB memory capacity) and less bandwidth for efficiency-oriented design.

While matrix operations can be accelerated using the concept of PIM, other types of computing, such as element-wise multiplication and activation functions, are not feasible to be implemented inside memory. Therefore, we implement several fixed function units with standard CMOS logic to ensure Flex-PIM has the flexibility to support various DNN models ranging from image classification networks, object detection models, to recurrent neural networks (RNNs).

As shown in Figure 5.12, multiplier array and adder array are used for element-wise multiplication and addition, respectively. Additionally, they, combined together, can be used to compute the Taylor series of some special functions such as sigmoid (σ). One should note that the design of multiplier and adder is implementation-dependent. For inference only implementation, fixed point multiplier/adder are enough while floating point unit (FPU) must be included if we want to perform training. Random number generator (RNG)

is used during training for weight/bias initialization (if no pre-trained model is available) and dropout layer. Pooling processor is used to perform average or max pooling and ReLU is for ReLU layer. There are several other fixed function units, such as max value search (used to find the largest exponent within an matrix, used for our PIM based floating point operation), and divider (used in batch normalization layer).

The last component in the Flex-PIM micro-architecture is the micro-processor which fetches/decodes the instructions and coordinates the data accessing and transmission.

5.7 Design Methodology

5.7.1 EDA Flow for Flex-PIM

As a memory-centric architecture, the hardware implementation of Flex-PIM presents as a big challenge due to the lack of proper memory compilers, especially for emerging non-volatile memories such as ReRAM and FeFET. Even for SRAM, conventional memory compilers can't be directly used here as we have very different peripherals and data access pattern. On the other hand, custom design is not practical either since this research is targeting to establish a flexible architecture design which should be orthogonal to memory techniques and reconfigurable towards various application scenarios.

To tackle this challenge and further enhance the flexibility of Flex-PIM, a design automation tool flow is developed which combins: (1) fully parameterized/synthesizable digital cores where the number of fixed function units, data bandwidth, and register size are controlled by parameters. (2) Auto-generated memory layout with different memory solutions, memory sub-array sizes, and crossbar aspect-ratios. (3) Script based synthesis/PNR flow (using Synopsys Design Compiler and Cadence Innovus) to perform manual floorplanning and layout generation. Figure 5.13 illustrates the design methodology and the tools/library we used. Essentially, given a design specification (such as memory solution, sub-array size, and system scale), the design flow will generate all the required information including area, power, timing, and layout automatically. The physical design based power,

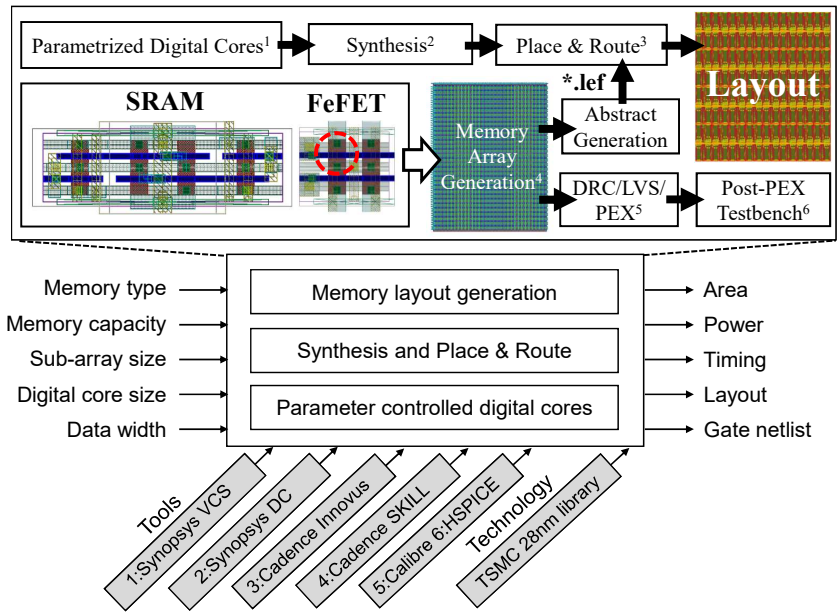


Figure 5.13: Flex-PIM design automation methodology.

timing, and area estimations are used to drive design space exploration as discussed next.

5.7.2 Design Space Exploration

A recent study shows that for a typical DNN model, more than 99% of computing is about matrix operations which can be accelerated inside memory array [123]. Therefore, to achieve the best performance, design space exploration is performed for the memory sub-arrays by altering the memory crossbar size and aspect-ratio leveraging the design automation methodology. One should note that a memory sub-array contains the memory crossbar and attached WL/BL peripherals. Different crossbar shapes lead to varying local registers, data bandwidth, and logic inside WL/BL peripherals.

As shown in Figure 5.14 (a) and (b), the power and area is plotted for 1 MB SRAM with different sub-array capacity and aspect-ratio (we put sub-arrays with same capacity in one group, for example, sub-arrays of 128×128, 64×256, 256×64 are put together). Figure 5.14 (c) shows the throughput for different configurations. The throughput number is calculated with how many output bits can be generated per clock cycle, which is the

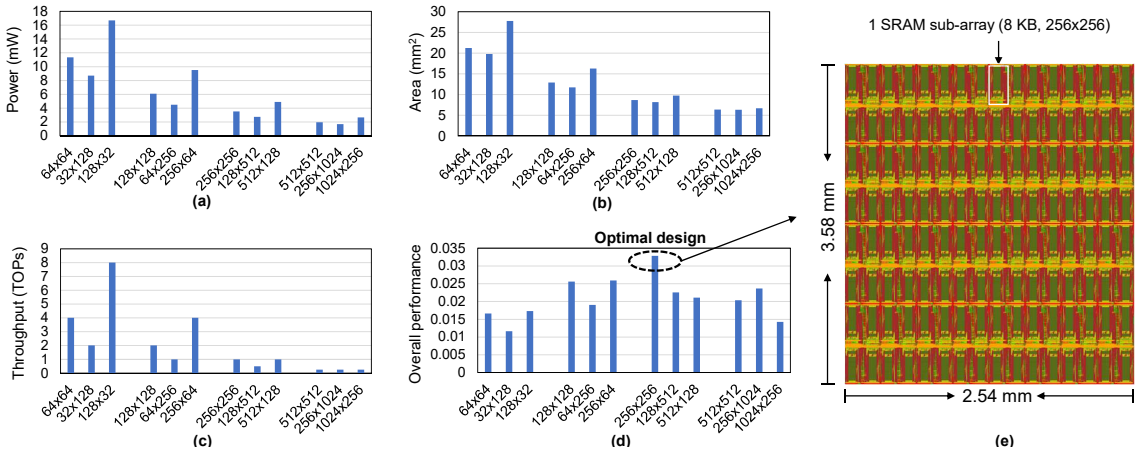


Figure 5.14: Design space exploration for SRAM in terms of (a) power, (b) area, (c) throughput, and (d) overall performance under different crossbar size with varying aspect-ratio. For example: 128×64 means the crossbar has 128 BLs and 64 WLs. (e) Layout view of 1 MB SRAM array (crossbar, peripherals, and H-NoC) used in Flex-PIM using TSMC 28nm technology.

number of total bitlines in the 1 MB SRAM (because each bitline SA outputs 1-bit in a cycle). Generally speaking, wider memory sub-arrays (i.e. number of BL is larger than number of WL) tend to have better throughput while taller memory sub-arrays (i.e. number of WL is larger than number of BL) have better power and area. For example, with the same capacity, 64×256 (64 BLs and 256 WLs) sub-array shows 2.1x better power efficiency, 1.2x less area, but 0.25x throughput compared with its 256×64 (256 BLs and 64 WLs) competitor.

To find the optimal design choice, we define another evaluation metric called **overall performance**, which is calculated by:

$$M = \frac{Throughput}{(Power \times Area)} \tag{5.5}$$

As shown in Figure 5.14 (d), larger arrays tend to have better overall performance than smaller arrays. For example, compared with 64×64 sub-array, 128×128 configuration is 1.6x better. However, it is noticed that for very large sub-arrays, the memory clock frequency must be reduced since the SA becomes slower due to the increasing WL/BL

	Design objective	Memory solution	Memory sub-array	Peak throughput	Memory bandwidth	Chip area	System power
HP_config	High performance	SRAM	2048 (16 MB)	32.76TOPs (Clk_mem: 4 GHz)	512 GB/s	123 mm^2	60.5 W
LP_config	Low power	FeFET	256 (2 MB)	2.05TOPs (Clk_mem: 2 GHz)	60 GB/s	6.2 mm^2	2.1 W

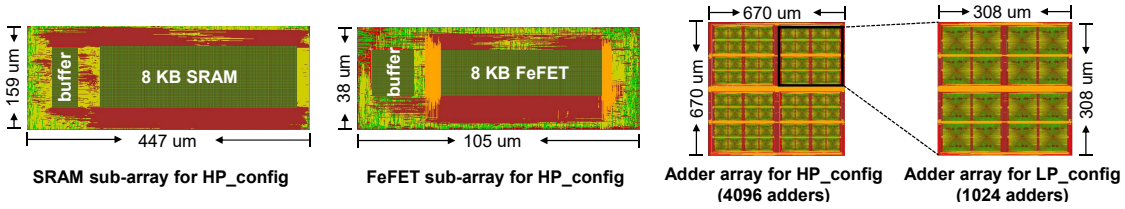


Figure 5.15: Two prototype implementations for Flex-PIM and layout views for sub-blocks.

parasitic (i.e. slower WL/BL charge and discharge). For example, with 512×512 configuration, the SRAM write time is 1.1 ns, limiting the clock frequency to 900 MHz, while for 128×128 configuration, the memory clock can safely run at 5 GHz.

Based on the simulation results, 256×256 sub-array size is used as the rule-of-thumb design, which is same with the SRAM in modern processor even though we have a drastically different WL/BL peripherals [29]. Figure 5.14 (e) shows the layout of 1 MB SRAM which contains 128 256×256 sub-arrays. The total area is $9.1mm^2$ including memory crossbar, peripherals, and H-NoC.

5.8 Experimental Results

The performance of Flex-PIM is simulated using a custom cycle-level simulator which is directly built on the software-hardware interface. The simulator analyzes the instructions and calculates the number of cycles for each operations as well as resource utilization (similar with the function of Flex-PIM micro-processor). The cycle time is calculated based on detailed circuit synthesis and PNR results (discussed in section 5.7). Ultimately, the application level performance evaluation is based on the coupling of timing analyses from cycle-accurate simulator and power modeling from circuit-level simulation.

Table 5.3: Benchmark DNN models.

DNN models	Application/Dataset	Parameter number	Complexity
AlexNet	Image classification (ImageNet)	61 M	0.7 GOPs
VGG-16	Image classification (ImageNet)	138 M	14.5 GOPs
ResNet-50	Image classification (ImageNet)	25.6 M	3.9 GOPs
LSTM	Activity classification (HAR)	2.3 M	0.32 GOPs
MobileNet-SSD	Object detection (MS-COCO)	6.8 M	1.2 GOPs

5.8.1 Two configurations

Flex-PIM is designed to be flexible and configurable to serve different application scenarios. This research experiments with two different Flex-PIM configurations. The first one is SRAM based performance oriented design (named **HP_config**) and the second one is FeFET based low power inference-only engine (named **LP_config**). The key design specifications are listed in Figure 5.15. For HP_config, SRAM is a preferred memory solution due to its fast read and write speed and non-disruptive sensing. For LP_config, both ReRAM and FeFET are suitable but we choose FeFET because it demonstrate lower read/write energy than ReRAM. While both configurations have 1 GHz system clock, the memory blocks are given a higher clock frequency (i.e. *Clk_mem*) for better performance. The off-chip memory bandwidth for HP_config and LP_config are set to be 512 GB/s (same with TPU-v2 [13]) and 60 GB/s (same with Nvidia Jetson TX2), respectively.

5.8.2 Performance analyses

As shown in Table 5.3, this work explores 5 different types of DNN models with varying parameter size and computing complexity (i.e. GOPs): Three (AlexNet [1], VGG-16 [2], and ResNet-50 [3]) for image classification using ImageNet dataset [23]; One (LSTM [39]) for human activity detection with UCI-HAR dataset [124]; And one (SSD [118]) for object detection with MS-COCO dataset [125].

It is observe that simpler/shallower neural networks (AlexNet and LSTM) tends to have

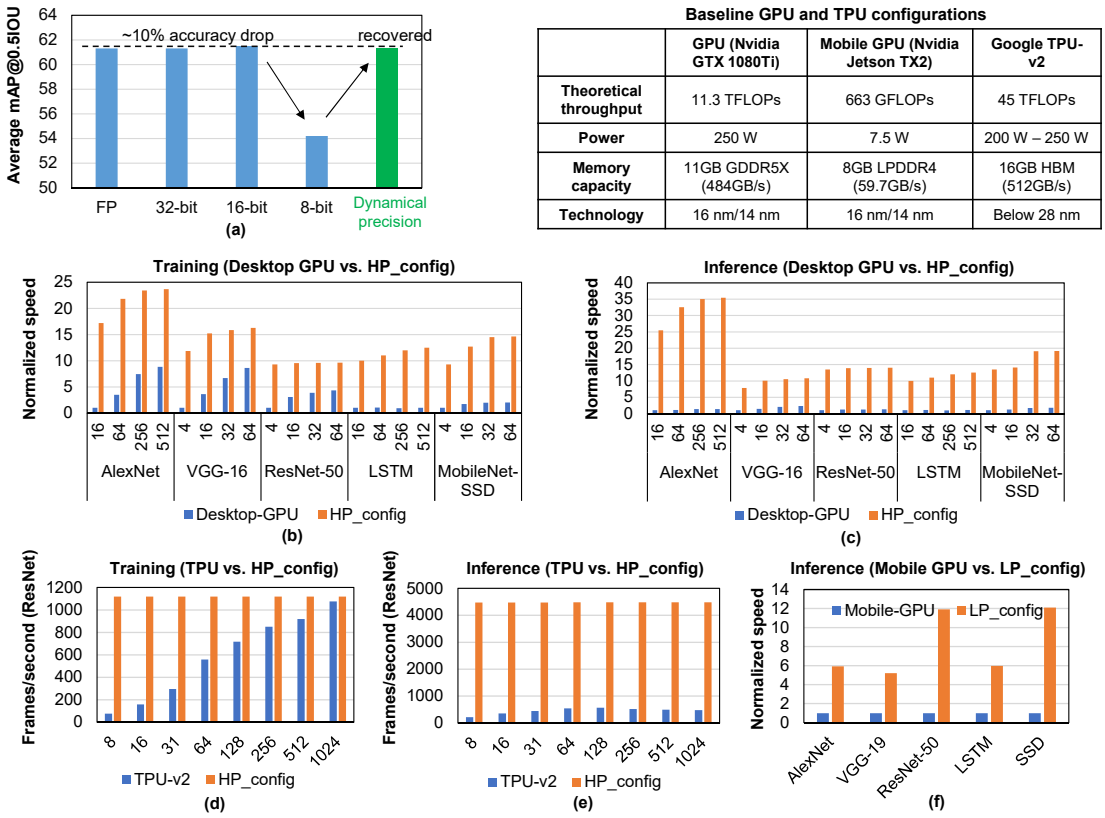


Figure 5.16: (a) Accuracy of SSD with different bit-precision. (b, c) Normalized training/inference speed of desktop GPU and Flex-PIM HP_config for DNN models with varying batch size. (d, e) Training/inference speed of TPU-v2 and Flex-PIM HP_config for ResNet-50 with varying batch size. (f) Normalized inference speed of mobile GPU and Flex-PIM LP_config with batch size 1.

better robustness towards quantization while the deeper networks (ResNet and VGG) are more sensitive. For example, with 8-bit precision (quantization and re-training), AlexNet shows almost similar accuracy compared with floating point but the accuracy drops more than 30% in the case of ResNet-50. Therefore, for AlexNet and LSTM, 8-bit is used for inference; for ResNet and VGG, 16-bit is used instead. The benefits of dynamical bit-precision using MobileNet-SSD is also evaluated. As shown in Figure 5.16 (a), with uniform 8-bit precision, the object detection rate drops by 10%. With dynamical-precision (8-bit for front-end feature extractor and 16-bit for back-end detector), the accuracy can be fully recovered. For training, 32-bit floating point is applied for all the benchmark networks.

The processing speed of HP_config is compared against the measured data from desktop GPU (Nvidia GTX 1080Ti) and Google TPU-v2. For LP_config, the data from a mobile GPU (Nvidia Jetson TX 2) is used as the baseline. The inserted table in Figure 5.16 shows the specifications for this baseline hardware accelerators. One should note that for TPU-v2, the detailed hardware information is still unraveled. The power is estimated based on a third-party anaylses [126]. Also, since one TPU node contains 4 TPU chips, the measured data are divided by 4.

Figure 5.16 (b) and (c) shows the speed comparison of desktop GPU and Flex-PIM HP_config for training and inference across the benchmark DNN models under varying batch sizes. For complex DNN models (such as VGG-16, ResNet-50, and SSD), smaller batch sizes is chosen to avoid the *run out of memory* error. Also, since models have very different performance number (i.e. frames/s), the data towards the GPU speed is normalized against minimum batch size for each benchmark to achieve better visualization. It is observed that HP_config outperforms GPU solution by 7.2x and 13.6x in terms of training and inference speed, respectively. Additionally, desktop GPU's power is 4.1x higher than Flex-PIM HP_config, resulting up to 56x computing efficiency (GOPs/W) improvement with our HP_config.

Figure 5.16 (d) and (e) shows the comparison between Flex-PIM HP_config and TPU-v2 using ResNet-50 with different batch sizes. The source code executed on TPU for ResNet-50 is provided by the TPU developing team and highly optimized to guarantee the performance. Since TPU-optimized source code for other DNN models are not available, custom models is developed and executed on TPU, with a less preferable (similar with desktop GPU, thus not shown here) performance. It is observed that TPU demonstrates good training performance when the batch size is large (similar performance with our HP_config when the batch size is 1024, which is the recommended number for TPU execution). It is also noticed that TPU shows less performance for inference than training, even though the reason is still not clear. On average, the HP_config outperforms TPU by 4.1x and 10.8x for

training and inference, respectively. In total, FlexPIM achieves up to 45x improvement in terms of GOPs/W.

The last comparison is between mobile GPU and Flex-PIM LP_config, shown in Figure 5.16 (f). Since they both target for low-power embedded platforms which require real-time processing. The batch size is set to 1 and only evaluate the inference speed. On average, LP_config achieves 8.2x speed up. Together with the 3.6x energy saving over the mobile GPU, Flex-PIM LP_config demonstrates 30x better computing efficiency.

5.9 Related Works

5.9.1 DNN quantization algorithms

Early stage of the research on DNN quantization algorithms suffers from large accuracy drop when testing on deep network models and large dataset. For example, TWN [127] loses 5% top-1 accuracy for AlexNet [1] on ImageNet [23] using 2-bit weight; XNOR-NET [21] and DoReFa [114] degrade the top-1 AlexNet ImageNet accuracy by 12% and 8% with binary weight and activation, respectively.

Recently, more advanced algorithms have been proposed to compress DNN with minimal accuracy degradation. Deep-compression [19] reduces the bit-precision of Conv layer and fully-connected layer (FC layer) to 8-bit and 5-bit, respectively. Combining with pruning, Deep-compression achieves negligible accuracy loss for AlexNet on ImageNet. Apprentice [22] leverages the knowledge distillation techniques and reduces the bit-precision of intermediate layer to 2-bit weight and 8-bit activation, resulting to 2.1% accuracy for ResNet on ImageNet. To guarantee accuracy, apprentices employs floating point precision for the first and last layers in its training pipeline. LQ-NETs employs an adaptive quantization strategy, shows 0.3% accuracy loss for ResNet with 4-bit weight and floating point activation. WRPN [128] exploits to compensate the quantization accuracy loss by doubling the channel size. Although WRPN achieved better compression rate and higher accuracy, it introduces additional computation with more convolutional channels. Several

other works, such as UNIQ [116], and PACT [117], also demonstrates good compression rate with insignificant accuracy drop. However, as mentioned before, most of them requires iterative/multi-pass training and additional trainable parameters.

Besides quantization, several other DNN model reduction techniques have been explored in the recent past. *Weight pruning* leverages the inherent redundancy in the number of weights in DNN models. During training, weight with small magnitude is pruned iteratively based on a threshold. While presenting as the very promising solution for DNN model reduction, its hardware implementation is very challenging due to the irregular sparsity and weight parameter indexing. A recent study shows that the computing efficiency will degrade unless the pruning can yield more than 60% compression rate [129]. *Knowledge distillation* reduces the DNN model depth using a teacher-student strategy. A deeper (more complex and larger) network is first trained. Then it is used to teach a smaller student network on the same task [22]. For example, the teacher network can be ResNet-50 and the student network is ResNet-18 [3]. *Hashing and weight sharing* is used to alias several weight parameters into few hash buckets, effectively lowering the parameter memory footprint [47].

5.9.2 DNN hardware accelerators

ASIC and FPGA: For logic-centric architecture, reducing the memory access is the most critical design objective to achieve high performance. For example, DaDianNao [10] and ShiDianNao [11] put DNN parameters inside on-chip eDRAM. Eyeriss [14] reduces the data movement by data reusing. ESE [15] is an FPGA based LSTM accelerator, optimized for sparse model which is compressed by pruning and quantization. TPU [130] is featured for its systolic matrix multiplication array. It also implements large on-chip memory (28 MB in TPU-v1) to reduce the DRAM access.

Near Memory Processing: Rather than reducing or eliminating DRAM access, near memory processing aims to reduce the off-chip memory access latency and improve the

Table 5.4: Performance comparison with other DNN accelerators.

	Technology	Hardware	Training support	Parameter storage	Power (W)	Area (mm^2)	Computing efficiency	Peak throughput
ESE	22 nm	FPGA	No	DRAM (off-chip)	41	N/A	6.88 GOPs/W	275 GOPs
DaDianNao	28 nm	ASIC	No	eDRAM (on-chip)	20.1	67.7	286 GOPs/W	5.7 TOPs
TPU-v2	--	ASIC	Yes	DRAM	~ 250	--	45 GOPs/W	11.3 TOPs
Deep train	15 ns	NMP	Yes	DRAM	7.2	--	566 GOPs/W	7.2 TOPs
ISAAC	28 nm	PIM	No	ReRAM	65.8	85.4	381 GOPs/W	25.1 TOPs
Neural Cache	28 nm	PIM	No	SRAM	52.9	--	529 GOPs/W	28 TOPs
FERA	28 nm	PIM	No	FeFET	2.3	13.14	443 GOPs/W	1.0 TOPs
HP_config	28 nm	PIM	Yes	SRAM	60.5	123	541 GOPs/W	32.8 TOPs
LP_config	28 nm	PIM	No	FeFET	2.1	6.2	976 GOPs/W	2.1 TOPs

bandwidth utilization. Neurocube [17] and Deep Train [18] propose a scalable architecture which vertically stacks the 3-D high bandwidth DRAM and logic tier together to achieve data access parallelism.

Processing-in-memory: PRIME [25] and ISAAC [26] are the pioneer works for PIM DNN architecture which use ReRAM as the memory candidate. They both employ the VMM-in-memory configuration and suffer from the overhead of analog/digital conversion. PipeLayer [27] optimizes the execution pipeline of in-memory computing and enables the in-situ training. FERA [30] using FeFET to replace ReRAM to further reduce the read/write energy. More recently, logic-in-memory based PIM configuration is investigated including DRISA [28], a DRAM based reconfigurable architecture and Neural Cache [29], a SRAM based bit-serial design.

A detailed comparison is performed between these DNN accelerators implemented with ASIC, FPGA, NMP, and PIM architecture. The key design features are summarized in Table 5.4. Beyond the flexibility and reconfigurability, Flex-PIM outperforms other PIM architectures by leveraging the following merits: (1) The key horse power of Flex-PIM comes from the all-digital VMM engine which eliminate the slow/power-hungry ADC, plus the high memory clock frequency. (2) Well-developed EDA flow for design space exploration and find the optimal design choice. (3) For FeFET based low power design, we also benefit from the succinct memory cell structure and ultra-low read/write energy.

CHAPTER 6

DESIGN OF RELIABLE DNN ACCELERATOR WITH UN-RELIABLE RERAM

6.1 The challenge of ReRAM's device variation

A major challenge for designing reliable ReRAM (and other types of emerging non-volatile memory techniques such as FeFET and STT-MRAM) based DNN accelerator is the inherent unreliability of the devices, i.e., the stochastic variations of ReRAM device resistance (variation exists in both high resistance state (HRS) and low resistance state (LRS)). Unlike the digital approach where the computation can be accurately performed as long as the bit-width is precise enough, the computation inside the ReRAM crossbar (i.e. multiplication-accumulation operation (MAC)) is executed in an analog fashion. Deviation of device resistance directly leads to errors in the sum-of-products result, thereby significantly degrading computing accuracy [36, 131]. The degradation can be largely alleviated with the all-digital PIM designs [50, 111], however, the accuracy still can not be guaranteed especially when the device on/off ratio is not large enough to ensure good read margin.

Recent papers have developed hardware-based techniques to improve robustness of DNN under ReRAM variations. For example, Chen et. al. presented a dual-reference multilevel sense amplifier (SA) to improve the sensing accuracy and ReRAM cell is used as binary device to diminish the overlapped region between two resistance levels [132]. Lin et. al. proposed a simulation framework to model the impact of noise to the accuracy of ReRAM based DNN accelerator and a workload-dependent sensing scheme is developed for better inference accuracy [133].

In this work, we proposed a complementary, algorithm-driven approach to design reliable DNN accelerator with unreliable ReRAM devices. The work makes following key contributions:

- We employ the dynamic fixed point (DFP) data representation [134] for mapping DNN weight matrices into the ReRAM crossbar during inference. In particular, we use different decimal point for each layers of neural network to maximize the utilization of ReRAM devices and reduce errors introduced by the unused most significant bits (MSBs).

- We propose a device-variation-aware (DVA) training methodology to enhance the robustness of neural network. By injecting random noise in the parameters during training, the trained model demonstrates high degree of resilience to parameter variation during evaluation stage.

We evaluate the proposed algorithms with CNNs (AlexNet and VGG) for image classification task and RNNs (simple RNN and stacked LSTM) for human activation recognition task. We observe that for all the benchmark applications, the computing accuracy improved by 24% on average across different levels of device variability (from 0% to 50%). The proposed algorithms also enhance the network robustness to input noise, making it suitable for deployment on low-power system with noisy sensors (such as low resolution camera). The proposed method can be used with any existing ReRAM based DNN accelerators with negligible hardware design overhead, and no inference speed/energy-efficiency drop.

6.2 Variability in ReRAM

The stochastic variation of ReRAM device resistance presents as a key challenge for the design of reliable ReRAM DNN accelerator. Recent study [135] shows that the variability is one of the intrinsic feature of ReRAM due to the stochastic nature of the generation and rupture of oxygen vacancies. Beside the device-level randomness, other factors such as fluctuation of res/reset voltage, noise introduced during fabrication can also contribution to the variation. Device measurements indicate that the resistance distribution of ReRAM (both HRS and LRS) follows normal distribution or log-normal distribution [133, 135].

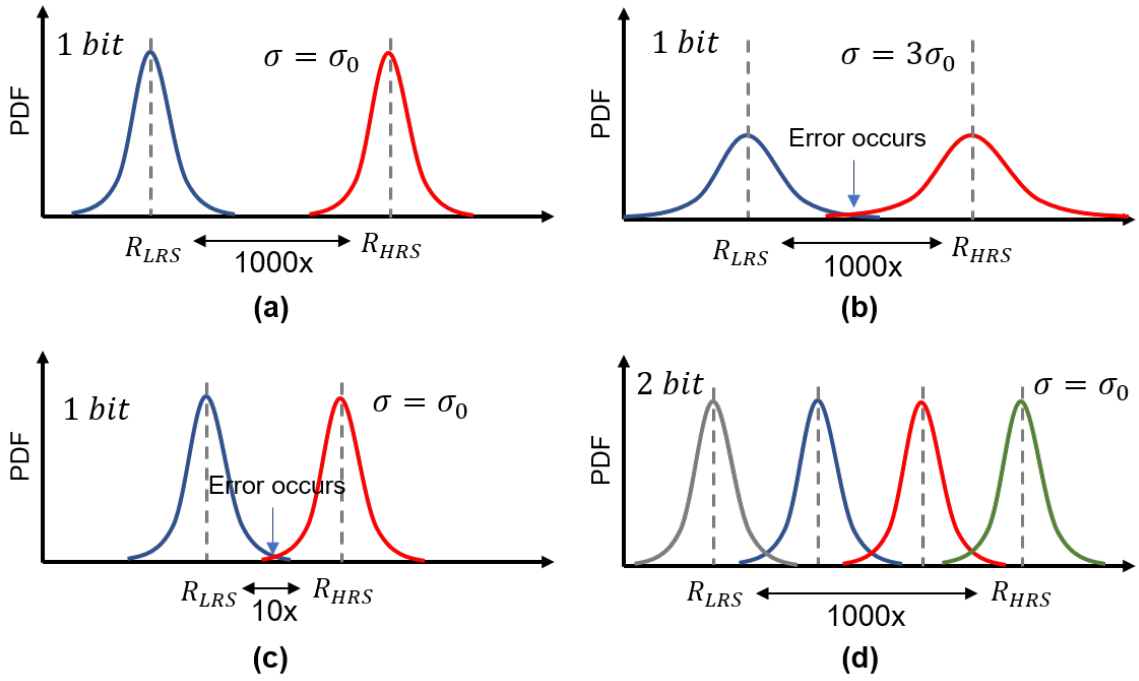


Figure 6.1: Factors that can affect the computing accuracy of ReRAM based DNN accelerator.

The impact of device variability to the computing accuracy is determined by three factors, namely, the deviation of the statistical distribution (σ), device on/off ratio ($r_{on/off}$) and how many bits to be stored in a cell. Figure 6.1(a) and (b) shows that large deviation reduces the read margin and error occurs when the device is programmed into the overlapped region. Similarly, as shown in Figure 6.1(c), small on/off ratio squeezes the margin between different resistance states. When the device is used as MLC (Figure 6.1(d)), neighbouring states tends to have larger overlap, making the computation more error-prone. Recent works [132, 36, 133] use ReRAM as binary device (i.e. 1bit per cell) to improve the computing reliability. We also consider using binary ReRAM based VMM engine for the rest of this chapter.

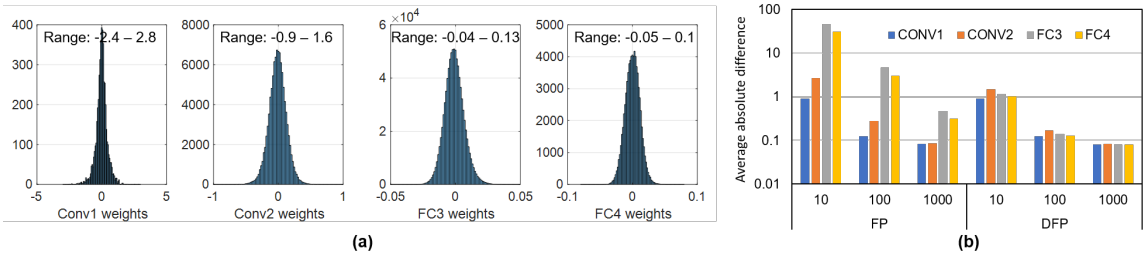


Figure 6.2: (a) Parameters distribution from different layers of AlexNet. (b) Parameters readout error caused by limited on/off device ratio ($R_H : R_L = 10, 100, 1000$) and resistance variation ($\sigma = 10\%$).

6.3 Proposed Methodology

6.3.1 Dynamical fixed point data representation

Dynamical fixed point (DFP) is a special data representation formation which allows us to adaptively change the location of the decimal point based on the range of data [134]. The concept of DFP is quite useful considering the fact that the range of parameters inside each layer of DNN can vary a lot[1]. As shown in Figure 6.2(a), we plot the weights distribution of AlexNet trained with CIFAR-10 dataset. The weight distribution in the first Conv layer is 10x larger than the later FC layers. With conventional fixed point data format, the variation in the first few MSBs (most significant bit) can be very detrimental for small parameter value especially in the case where the device on/off ratio is not large enough.

For example, using conventional fixed point to accommodate the weights of Alexnet, we need 1 bit for sign, 2 bits for integer and 5 bits for fraction. Let's say we chose one parameter ($\hat{A} = 0.125$) and program it to ReRAM devices, as in Figure 6.3. The stored weight would be identical to $\hat{A}$ if the device is ideal (i.e. resistance on/off ratio $R_H : R_L = \infty$ and resistance shift $\Delta = 0$). However, if the device has limited on/off ratio (i.e. $R_H : R_L = 10$), a large readout error can be observed due to the non-zero current read from MSBs. The readout value can be further disturbed if we consider the device resistance shift

[1]Should note that the highly biased weight distribution after training is largely caused by the gradient backpropagation. In most recent DNN configurations, with the Batch Normalization, this phenomenon is much less significant. More details about the benefits of Batch Norm can refer to [136].

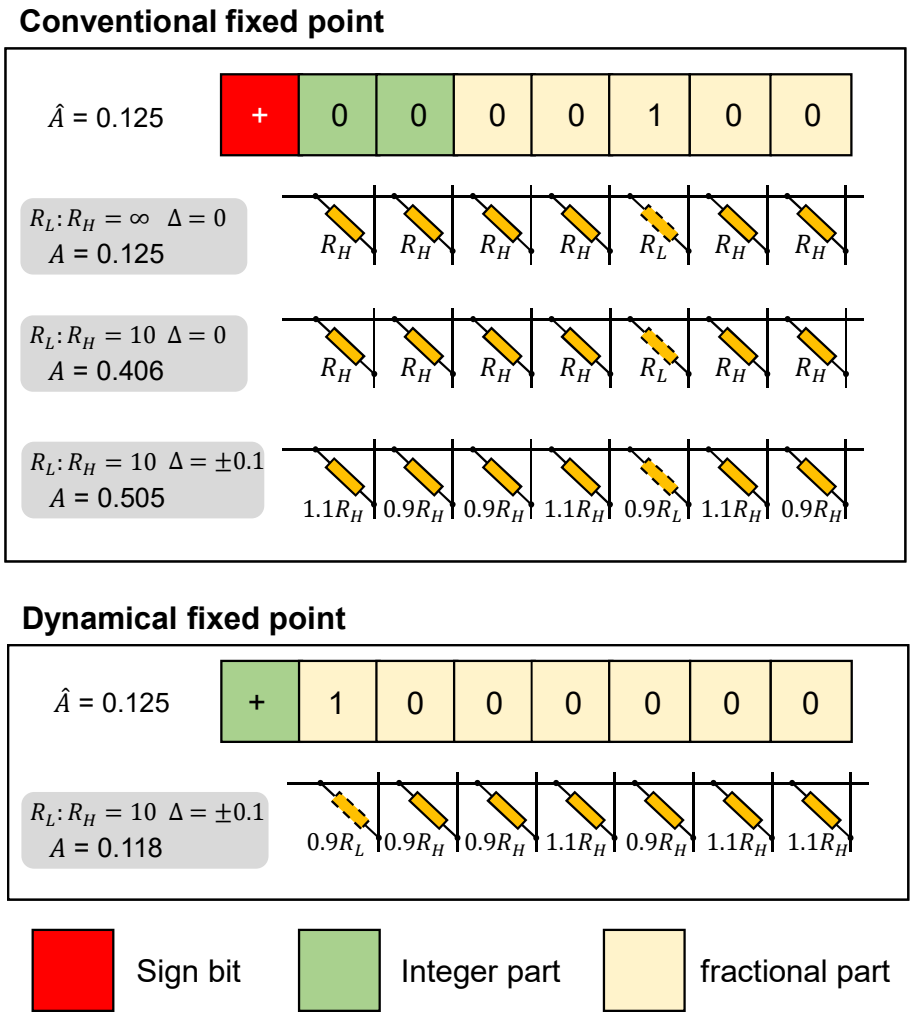


Figure 6.3: Convention fixed point v.s. DFP.

(i.e. $R_H : R_L = 10$ and $\Delta = 0.1$). As illustrated in the second part of Figure 6.3, the final readout from a realistic ReRAM crossbar ($A = 0.505$) skews more than 300% with conventional fixed point data representation.

On the other hand, with DFP, we can left shift the decimal point position to make sure there is no unused MSBs, significantly reducing the readout error. As shown in Figure 6.3, with DPF, the readout from ReRAM crossbar only shows 7% shew over the original weight value.

We also evaluate the benefits of DFP under different on/off ratio statistically, as demonstrated in Figure 6.2(b), suggesting DFP together with higher on/off ratio can help to re-

duce the readout error caused by device variability. For the rest of the paper, we assume the on/off ratio is 1000, which is a realistic number from recent measurement data [135].

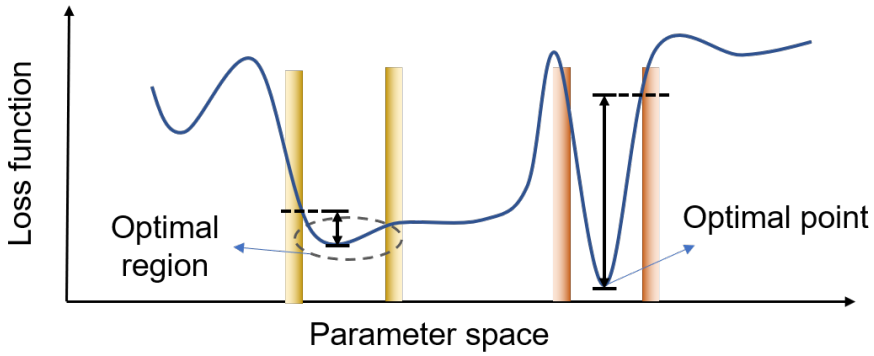
We should note that prior works have demonstrated that dynamic fixed-point representation is beneficial to speed up the training/inference of machine learning applications [134]. In this work, rather than accelerating DNN computing, we exploit the dynamic fixed point as a technique to reduce the impact of device variation.

6.3.2 Device-Variation-Aware Training

The training of DNN is a process to find the **optimal point** for the loss function (also called cost function) inside the parameter space. However, as shown in the top of Figure 6.4, a small skew from the optimal point leads to a huge increase for the loss function, indicating the DNN accuracy is sensitive to parameter variation. On the other hand, if there is an **optimal region** in which the loss function is relatively small (may not achieve the global minimum) everywhere, we can then use the parameters inside this region to establish a noise robust network.

This inspires us to implement a device-variation-aware (DVA) training methodology where we intentionally add noise to DNN parameters during training to improve the robustness. The pseudo code in Figure 6.4 illustrates how we add noise to convolution layer. For each training batch, we randomly generate a noise matrix with the same size of the convolution kernel specifying the mean and deviation. Then we add the noise to the parameters by element-wise multiplication and use the noised kernel for the rest operations. Similar procedure is implemented for fully-connected layer and computation inside basic RNN/LSTM.

We have developed a noise model to correlate the device variability with the parameter noise while considering the bit-level representation of the parameters analytically. We first assume the device variation follows Gaussian distribution and the standard deviation is




```
Def Convolution_layer(input, kernel, mean, stddev):
        noise = normal_distribution(mean, stddev)
        noised_kernel = elementwise_multiplication(kernel, noise)
        conv = conv2d(input, noised_kernel, stride, padding)
        conv = add_bias(conv, bias)
        conv = relu(conv)
        Return conv
```

Figure 6.4: **Top**: The comparison between optimal point and optimal region of the loss function in parameter space. **Bottom**: Pseudo code for DVA training process.

proportional to the mean value:

$$X \sim N(\mu, \sigma^2 = (\gamma\mu)^2) \tag{6.1}$$

where γ is a constant coefficient reflecting device noise level. Assuming the DNN parameter is 4-bit number ($w_3 w_2 w_1 w_0$), the target readout value would be:

$$S = 8w_3 + 4w_2 + 2w_1 + w_0 \tag{6.2}$$

Since each device follows an independent normal distribution, the distribution for the sum (S) is derived as:

$$S \sim N(8w_3 + 4w_2 + 2w_1 + w_0, \sigma^2_{parameter}) \tag{6.3}$$

$$\sigma^2_{parameter} = \gamma^2[(8w_3)^2 + (4w_2)^2 + (2w_1)^2 + (w_0)^2] \tag{6.4}$$

This analytic model helps us to statistically characterize the DNN parameter noise distribution given the device variation. For example, with device deviation coefficient $\gamma = 10\%$, the DNN parameter distribution roughly follows a normal distribution with $\sigma_P = 8\%P$ where P is the value of parameters.

In summary, if the variability in the ReRAM process is known, we develop the corresponding noise model and use that to introduce noise during DNN training. Therefore, the trained DNN become inherently aware of device variation. Besides, in the following section we also study the effect of mismatch between (i) the actual ReRAM variation experienced during inference and (ii) the ReRAM variation assumed during training.

6.4 Simulation Results

The proposed algorithms are evaluated with 4 benchmarks, including AlexNet and VGG for image classification (CIFAR-10 dataset), basic RNN and LSTM for human activity recognition (UCI-HAR). All DNN models are implemented with Tensorflow machine learning framework and runs on a Nvidia GTX-1080Ti GPU. Recent study shows that ReRAM can achieve 1000x on/off ratio and the device variation (σ) varies from 5% to 50% of the resistance value [135]. The simulation is configured based on these observations and ReRAM device is treated as a binary cell.

6.4.1 Variability Simulation

Since a device only store one bit for a parameter, the device variation can't be directly interpreted as the noise of DNN parameters. The first part of Figure 6.5 shows the computation flow using ReRAM crossbar as the computing engine. We first need to convert the decimal parameter matrix ($P_{decimal}$) to a corresponding binary format (P_{binary}) and program each

bit to an ReRAM device. Noise is added to the programmed value based on the device variability. Then we sum up the noised binary matrix ($P_{binary-noised}$) column-wisely, quantize with ADC, and shift&add the partial results together. A Monte-Carlo analysis of the preceding flow can be performed using circuit simulators to model the effect of ReRAM variation to MAC outputs.

However, the detail approach discussed above, although accurate, but computationally in-efficient when implemented in a DNN software simulation platform. Therefore, as shown in the later part of Figure 6.5, we propose to directly convert the noisy binary weight matrix ($P_{binary-noise}$) back to its decimal format ($P_{decimal-noise}$) and run simulation using existing software framework. One should note that these two approaches are essentially identical if the ADC is ideal. Our approach is still a good approximation even with non-ideal ADC. For example, assuming 8-bit ADC for a crossbar with 64 rows and device variation is 0.2, we randomly chose a matrix (programmed into ReRAM crossbar) and vector (used as input) to perform MAC operation. The output result after normalization for MAC operation is 1.0, 1.0146, and 1.0147 for accurate solution, detailed evaluation flow (top part of Figure 6.5), and our approximation (bottom part of Figure 6.5), respectively. This indicates the mismatch between our approximation and the detailed simulation is negligible comparing with the error introduced by device variation.

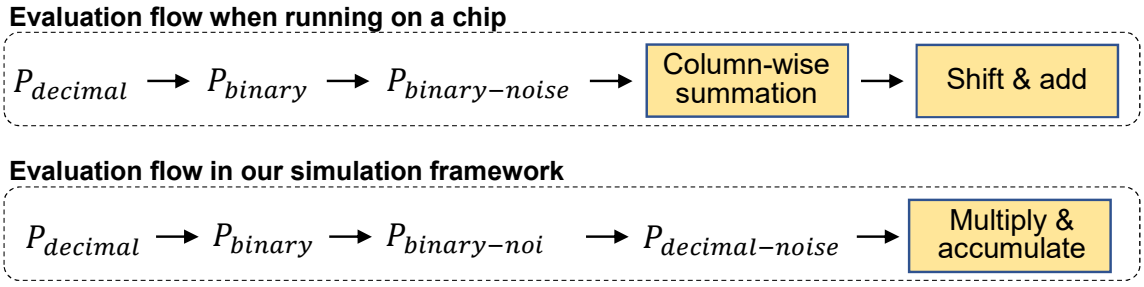


Figure 6.5: Evaluation flow when running on an ReRAM chip v.s. Evaluation flow in our simulation framework.

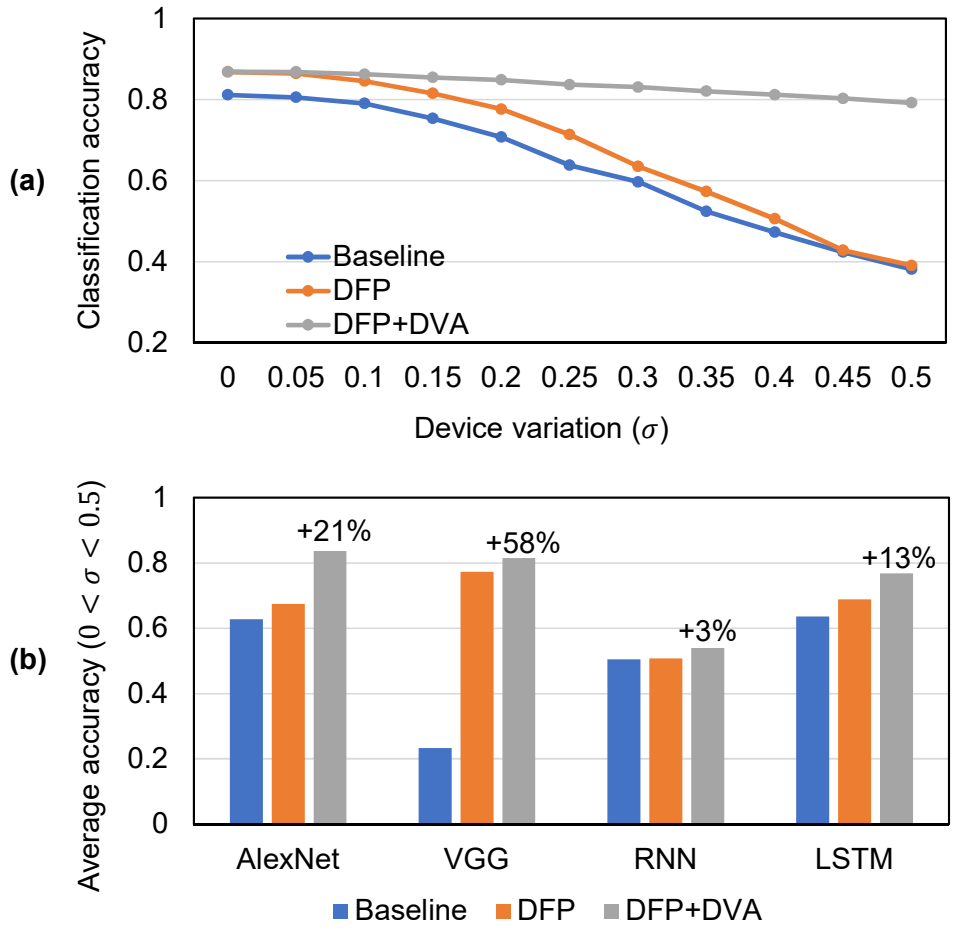


Figure 6.6: Accuracy analysis: (a) Inference accuracy of AlexNet with baseline, DFP, and DFP+DVA configurations and (b) Average accuracy improvement across different device variation level for AlexNet, VGG, basic RNN, and LSTM.

6.4.2 Accuracy Improvement under device variability

As shown in Figure 6.6(a), the classification accuracy of AlexNet under different device variation for the baseline (i.e. no optimization), DFP enabled, and DFP+DVA enabled configurations is plotted. The benefits of DFP is more notable when the device variation is relatively small since the error caused by unused MSBs is dominating. With higher device variation, both baseline and DFP configuration show large accuracy drop, indicating the network is sensitive to parameter noise. The DFP+DVA approach demonstrates the most promising results with less than 7% accuracy drop even under 50% device variation. Similar observations are made in other networks (VGG, basic RNN, and LSTM). As shown

in Figure 6.6(b), we plot the average accuracy across varying device variation (from 0 to 50%) for all four benchmark networks. On average, 24% accuracy improvement is achieved. One should note that for basic RNN, there is only one weight matrix, thus only one decimal point globally. Therefore, DFP configuration is essentially the same as the baseline. We also note that the improvement for simple network (such as the basic RNN) tends to be smaller. This is because with less parameters and simpler structure, the network naturally shows better robustness compared with larger and deeper counterparts.

Noise applied during DVA training

Device variation (σ)	0.00	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50
0.00	0.868	0.872	0.868	0.866	0.863	0.856	0.848	0.846	0.844	0.833	0.826
0.05	0.864	0.869	0.869	0.865	0.862	0.855	0.850	0.845	0.843	0.836	0.826
0.10	0.849	0.859	0.863	0.863	0.860	0.854	0.850	0.847	0.842	0.834	0.827
0.15	0.818	0.838	0.853	0.857	0.858	0.853	0.850	0.845	0.842	0.835	0.828
0.20	0.772	0.812	0.834	0.847	0.852	0.845	0.847	0.846	0.842	0.835	0.827
0.25	0.720	0.774	0.810	0.833	0.841	0.841	0.842	0.842	0.841	0.832	0.825
0.30	0.661	0.725	0.780	0.809	0.828	0.829	0.836	0.836	0.837	0.830	0.826
0.35	0.598	0.664	0.735	0.784	0.805	0.815	0.825	0.830	0.831	0.827	0.823
0.40	0.528	0.606	0.688	0.754	0.783	0.795	0.814	0.819	0.823	0.819	0.818
0.45	0.470	0.544	0.637	0.719	0.753	0.772	0.797	0.805	0.815	0.812	0.810
0.50	0.409	0.488	0.577	0.669	0.715	0.746	0.779	0.790	0.801	0.802	0.807

Figure 6.7: Classification accuracy of AlexNet under the mismatch between noise used during training and variation experienced during evaluation.

6.4.3 Variation Mismatch between Training and Inference

The DNN model accuracy when device variation assumed during training differs from the actual variation during inference is also investigated, for example, due to lack of exact knowledge of the device characterization or device variation changes over time (aging or temperature changes), as shown in Figure 6.7. With small training noise, the accuracy is approaching the ideal number (i.e. AlexNet trained on GPU has 86.8% accuracy) when the evaluation noise is small (top-left corner) but drops a lot once the evaluation variation is large (bottom-left corner), implying the network has less robustness to large variation. On

the other hand, with large training noise, we observe there is a slight accuracy drop under zero evaluation noise (top-right corner) but the network is much more robust: only 2% drop when the evaluation noise increase from 0% to 50% (bottom-right corner). Therefore, if exact variation information is not available, we suggest training with a larger noise to improve reliability.

6.4.4 Impact of Non-Normal Variation

As mentioned previously, the device variation can also be modeled with log-normal [135] device variation. Accordingly, the DVA algorithms employing log-normal noise during training stage is explored. Table 6.1 shows that DFP+DVA produces the best performance and robustness to device noise with 15% accuracy improvement over the baseline approach. Interestingly, the baseline shows slightly better results comparing with the case when device variation follows normal distribution (as shown in Figure 6.6(a)). While there is no rigorous theoretical supports, we believe this is because the log-normal distribution has a non-symmetric probability distribution function (PDF) which tends to produces more small variables than normal distribution.

6.4.5 Improving the robustness to input noise

We study whether DFP+DVA can improve robustness to noise applied to the input data during inference. We introduce random noise with normal distribution to the image and run inference with models (AlexNet) trained with different parameter noise. As in Figure 6.8, the accuracy drops quickly with larger input image noise without DVA training. Much better robustness is observed when a DVA trained model is used for inference. Similar with

Table 6.1: Accuarcy of baseline, DFP, and DFP+DVA under log-normal distribution.

	0.0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**Average**
Baseline	0.85	0.83	0.76	0.69	0.57	0.39	0.67
DFP	0.86	0.84	0.79	0.75	0.60	0.40	0.72
DFP+DVA	0.86	0.84	0.83	0.81	0.79	0.73	0.82

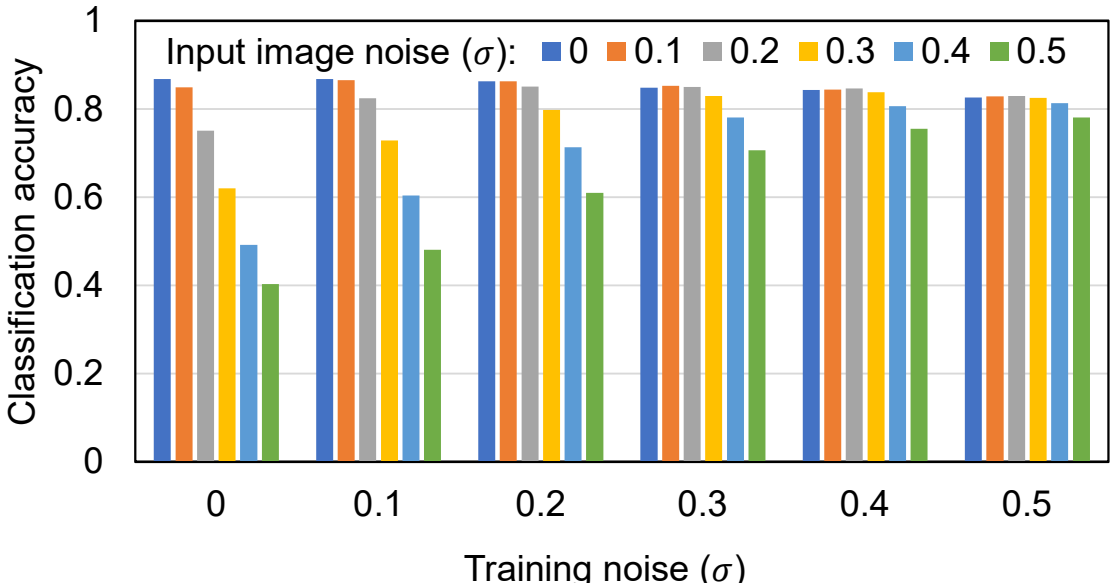


Figure 6.8: AlexNet classification accuracy with varying input image noise using trained models with different level of training noise.

the case in Figure 6.7, DVA enhance the robustness but slight scarifies the accuracy (less than 4% in the case of AlexNet) for clean images. The robustness against input noise is a major advantage for deployment of DNN in power-constrained platforms for real-world applications, for example, to tolerate noise introduced by low-power and low-cost cameras.

6.5 Brief Summary

We propose algorithmic approaches to design a reliable DNN accelerator with unreliable ReRAM devices. Our approaches are based on the dynamical fixed point data representation and device variation aware training. The experimental results demonstrate that the proposed approach not only enhance the network robustness under device variation, but also improves the accuracy when input contains noise.

CHAPTER 7
CONCLUSION

7.1 Key innovations of this dissertation

This research presents a software-hardware co-design approach to enhance the processing-in-memory DNN accelerator computing efficiency and accuracy. From the software perspective, a noise-robust DNN training algorithm is presented and a novel evolution algorithm based DNN layer-wise quantization method is proposed. On the other side, our hardware implementation employs a cross-cutting solution which integrates micro-architecture innovation, circuit optimization, and novel device exploration.

Chapter 3 presents a ReRAM based accelerator for RNN computation which is capable for the acceleration of various RNN computation including the basic RNN, LSTM, and GRU. Beyong the ReRAM based VMM engine for vector-matrix multiplication, dedicated special function unit is integrated to the design to support the calculation for activation functions. Dataflow is optimized to enhance the computing efficiency.

Chapter 4 proposes FERA, a FeFET based scalable architecture to accelerate DNN computation. With a cross-cutting solution combining emerging device technologies, circuit optimization, and micro-architectural innovations, state-of-the-art performance is achieved. A dedicated NoC design is proposed to provide solution for input data sharing and partial results summation.

Chapter 5 proposes Flex-PIM, an all-digital, fully synthesizable PIM architecture which supports dynamic computing precision. Genetic algorithm based layer-wise DNN quantization algorithm is proposed, making it possible to fully utilize the computing power of flexible precision hardware accelerator. Together with the developed Flex-PIM instruction set and software-hardware interface, this work makes PIM architecture much more flexible

and mature, paving the way to a real fabrication, which is also our future direction.

Chapter 6 presents algorithmic approaches to design a reliable DNN accelerator with unreliable ReRAM devices. Two algorithms are proposed in this work. The first is based on the dynamical fixed point data representation to reduce the error introduced by un-used MSB (i.e. most significant bit). The second algorithm, noise-aware training, enhances the DNN robustness towards the variation inside weight parameters.

7.2 Future work

While PIM based machine learning accelerators have demonstrated unprecedented efficiency over its digital logic counterpart, PIM architecture exploration is still in a very early stage and far less than mature. There are lots of open questions, obstacles, and challenges, once solved, can greatly enhance PIM's potential and stimulate PIM's wide adoption. From device perspective, various emerging memory techniques have been explored and optimized, but these memory techniques suffer from the unsatisfied uniformity, high read or write power, and low yield. The device with ideal characteristic for PIM is still missing and highly desired. From computing methodology perspective, analog PIM is advantageous for its large parallelism but suffers from the limited accuracy. On the other side, digital PIM is more promising for high-precision computing but has low computing density. Ultimately, the design philosophy (digital versus analog) might be application dependent with a vague boundary (or even reconfigurable) between different approaches. Finally, in terms of algorithms, with more research efforts devoted to the meta-learning (i.e. autoML), neural architecture search (NAS) as well as software/hardware co-design, machine learning based accelerator architecture search presents as a very promising future research direction.

Appendices

APPENDIX A

PUBLICATION LIST

[1] **Yun Long**, and Saibal Mukhopadhyay, "ReRAM Crossbar based Cellular Neural Network for High Efficient Computing," Non-Volatile Memory Workshop (NVMW 2016).

[2] **Yun Long**, Eui Min Jung, Jaeha Kung, and Saibal Mukhopadhyay, "ReRAM Crossbar based Recurrent Neural Network for Human Activity Detection," International Jointed Conference on Neural Networks (IJCNN 2016).

[3] Jaeha Kung, **Yun Long**, and Saibal Mukhopadhyay, "An Energy-Efficient Physical Platform for Solving Differential Equations," Post-Moore's Era Supercomputing Workshop (PMES 2016).

[4] Jaeha Kung, **Yun Long**, and Saibal Mukhopadhyay, "An Energy-Efficient Physical Platform for Solving Differential Equations," 42th Annual Government Microcircuit Applications Critical Technology Conference (GOMACTech 2017).

[5] Jaeha Kung, **Yun Long**, Duckhwan Kim, and Saibal Mukhopadhyay, "A Programmable Hardware Accelerator for Simulating Dynamical Systems," 44th International Symposium on Computer Architecture (ISCA 2017).

[6] (Invited) Jong Hwan Ko, **Yun Long**, Mohammad Faisal Amir, Duckhwan Kim, Jaeha Kung, Taesik Na, Amit Trivedi, and Saibal Mukhopadhyay, "Energy-Efficient Neural Image Processing for Internet-of-Things Edge Devices", 60th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS 2017).

[7] **Yun Long**, She Xueyuan, and Saibal Mukhopadhyay, "Accelerating Biophysical

Neural Network Simulation with Region of Interest based Approximation", 18th Design, Automation and Test in Europe (DATE 2018).

[8] **Yun Long**, and Saibal Mukhopadhyay, "ReRAM based Processing-in-memory Architecture for Recurrent Neural Network Acceleration", IEEE Transactions on Very Large Scale Integration Systems (IEEE TVLSI).

[9] **Yun Long**, Taesik Na, Rao Karthik, Sudhakar Yalamanchili, and Saibal Mukhopadhyay, "FERA: A Ferroelectric FET based Scalable Architecture for Data-intensive Computing", 37th International Conference on Computer Aided Design (ICCAD 2018).

[10] **Yun Long**, Xueyuan She, and Saibal Mukhopadhyay, "HybridNet: Integrating Model-based and Data-driven Learning to Predict Evolution of Dynamical Systems", Conference on Robot Learning (CORL 2018).

[11] Xueyuan She, **Yun Long**, and Saibal Mukhopadhyay, "Fast and Low-Precision Learning in GPU-Accelerated Spiking Neural Network", 19th Design, Automation and Test in Europe (DATE 2019).

[12] **Yun Long**, and Saibal Mukhopadhyay, "Design of Reliable DNN accelerator with Un-reliable ReRAM", 19th Design, Automation and Test in Europe (DATE 2019).

[13] BA Mudassar, Priyabrata Saha, **Yun Long**, MF Amir, E Gebhardt, T Na, JH Ko, M Wolf, and Saibal Mukhopadhyay, "A Camera with Brain–Embedding Machine Learning in 3D Sensors", 19th Design, Automation and Test in Europe (DATE 2019).

[14] **Yun Long**, and Saibal Mukhopadhyay, "ReRAM based Processing-in-memory Architecture for Recurrent Neural Network Acceleration", IEEE Journal on Exploratory Solid-State Computational Devices and Circuits (IEEE JXCDC).

[15] Priyabrata Saha, **Yun Long**, and Saibal Mukhopadhyay, "MagNet: Discovering Multi-agent Interaction Dynamics using Neural Network", Submitted to NeurIPS 2019.

[16] Xueyuan She, **Yun Long**, and Saibal Mukhopadhyay, "Deep Learning with Spike-assisted Contextual Information Extraction ", Submitted to NeurIPS 2019.

[17] BA Mudassar, Priyabrata Saha, **Yun Long**, MF Amir, E Gebhardt, T Na, JH Ko, M Wolf, and Saibal Mukhopadhyay , "CAMEL: An Adaptive Camera with Embedded Machine Learning Based Sensor Parameter Control", IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS), accepted for publication.

[18] Saibal Mukhopadhyay, **Yun Long**, etc., "Heterogenous Integration for Artificial Intelligence: Challenges and Opportunities", IBM Journal of Research and Development (IBM Journal) under review.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[6] Y. Long, X. She, and S. Mukhopadhyay, "Hybridnet: Integrating model-based and data-driven learning to predict evolution of dynamical systems," *arXiv preprint arXiv:1806.07439*, 2018.

[7] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[8] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[9] N. Brown and T. Sandholm, "Superhuman ai for multiplayer poker," *Science*, eaay2400, 2019.

[10] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, 2014, pp. 609–622.

[11] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 43, 2015, pp. 92–104.

[12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2017, pp. 1–12.

[13] *Google tpu-v2*, `https://cloud.google.com/tpu/docs/system-architecture`, Accessed: 2010-09-30.

[14] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, IEEE Press, vol. 44, 2016, pp. 367–379.

[15] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ACM, 2017, pp. 75–84.

[16] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, IEEE, 2016, pp. 243–254.

[17] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, IEEE, 2016, pp. 380–392.

[18] D. Kim, T. Na, S. Yalamanchili, and S. Mukhopadhyay, "Deeptrain: A programmable embedded platform for training deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2360–2370, 2018.

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[20] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 2019, pp. 925–938.

[21] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, Springer, 2016, pp. 525–542.

[22] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," *arXiv preprint arXiv:1711.05852*, 2017.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, Ieee, 2009, pp. 248–255.

[24] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-memory for energy-efficient neural network training: A heterogeneous approach," MICRO, 2018.

[25] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, IEEE Press, vol. 44, 2016, pp. 27–39.

[26] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[27] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, IEEE, 2017, pp. 541–552.

[28] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2017, pp. 288–301.

[29] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," *arXiv preprint arXiv:1805.03718*, 2018.

[30] Y. Long, T. Na, P. Rastogi, K. Rao, A. I. Khan, S. Yalamanchili, and S. Mukhopadhyay, "A ferroelectric fet based power-efficient architecture for data-intensive computing," in *Proceedings of the International Conference on Computer-Aided Design*, ACM, 2018, p. 32.

[31] Y. Ji, Y. Zhang, X. Xie, S. Li, P. Wang, X. Hu, Y. Zhang, and Y. Xie, "Fpsa: A full system stack solution for reconfigurable reram-based nn accelerator architecture," in *Proceedings of the Twenty-Fourth International Conference on Architec-*

tural Support for Programming Languages and Operating Systems, ACM, 2019, pp. 733–747.

[32] S Park, A Sheri, J Kim, J Noh, J Jang, M Jeon, B Lee, B. Lee, B. Lee, and H Hwang, "Neuromorphic speech systems using advanced reram-based synapse," in *2013 IEEE International Electron Devices Meeting*, IEEE, 2013, pp. 25–6.

[33] C Nail, G Molas, P Blaise, G Piccolboni, B Sklenard, C Cagli, M Bernard, A Roule, M Azzaz, E Vianello, *et al.*, "Understanding rram endurance, retention and window margin trade-off using experimental results and simulations," in *2016 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2016, pp. 4–5.

[34] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H.-S. P. Wong, "A neuromorphic visual system using rram synaptic devices with sub-pj energy and tolerance to variability: Experimental characterization and large-scale modeling," in *2012 International Electron Devices Meeting*, IEEE, 2012, pp. 10–4.

[35] S. Yu, Z. Li, P.-Y. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian, "Binary neural network with 16 mb rram macro chip for classification and online training," in *2016 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2016, pp. 16–2.

[36] Y. Long *et al.*, "ReRAM-Based Processing-in-Memory Architecture for Recurrent Neural Network Acceleration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 99, 2018.

[37] Y. Long, X. She, and S. Mukhopadhyay, "Hybridnet: Integrating model-based and data-driven learning to predict evolution of dynamical systems," *arXiv preprint arXiv: 1806.07439*, 2018.

[38] Y. Bengio *et al.*, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[39] Y. Long, T. Na, and S. Mukhopadhyay, "Reram-based processing-in-memory architecture for recurrent neural network acceleration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 99, pp. 1–14, 2018.

[40] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[41] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[42] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[43] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[44] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[45] X. Dai, H. Yin, and N. Jha, "Nest: A neural network synthesis tool based on a grow-and-prune paradigm," *IEEE Transactions on Computers*, 2019.

[46] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, 2016, pp. 1379–1387.

[47] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015, pp. 2285–2294.

[48] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "Cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[49] C. Li, Y. Yang, M. Feng, S. Chakradhar, and H. Zhou, "Optimizing memory efficiency for deep convolutional neural networks on gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16, Salt Lake City, Utah: IEEE Press, 2016, 54:1–54:12, ISBN: 978-1-4673-8815-3.

[50] Y. Long, D. Kim, E. Lee, P. Saha, B. A. Mudassar, X. She, A. I. Khan, and S. Mukhopadhyay, "A ferroelectric fet based processing-in-memory architecture for dnn acceleration," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 2019.

[51] B. Gao, Y. Bi, H.-Y. Chen, R. Liu, P. Huang, B. Chen, L. Liu, X. Liu, S. Yu, H.-S. P. Wong, *et al.*, "Ultra-low-energy three-dimensional oxide-based electronic synapses for implementation of robust high-accuracy neuromorphic computation systems," *ACS nano*, vol. 8, no. 7, pp. 6998–7004, 2014.

[52] P Huang, X. Liu, W. Li, Y. Deng, B Chen, Y Lu, B Gao, L Zeng, K. Wei, G Du, *et al.*, "A physical based analytic model of rram operation for circuit simulation," in *Electron Devices Meeting (IEDM), 2012 IEEE International*, IEEE, 2012, pp. 26–6.

[53] B Gao, S. Yu, N Xu, L. Liu, B Sun, X. Liu, R. Han, J. Kang, B Yu, and Y. Wang, "Oxide-based rram switching mechanism: A new ion-transport-recombination model," in *2008 IEEE International Electron Devices Meeting*, IEEE, 2008, pp. 1–4.

[54] U Russo, D. Ielmini, C. Cagli, A. L. Lacaita, S. Spiga, C Wiemer, M Perego, and M Fanciulli, "Conductive-filament switching analysis and self-accelerated thermal dissolution model for reset in nio-based rram," in *2007 IEEE International Electron Devices Meeting*, IEEE, 2007, pp. 775–778.

[55] S Park, H Kim, M Choo, J Noh, A Sheri, S Jung, K Seo, J Park, S Kim, W Lee, *et al.*, "Rram-based synapse for neuromorphic system with pattern recognition function," in *Electron Devices Meeting (IEDM), 2012 IEEE International*, IEEE, 2012, pp. 10–2.

[56] X. Guan, S. Yu, and H.-S. P. Wong, "A spice compact model of metal oxide resistive switching memory with variations," *IEEE electron device letters*, vol. 33, no. 10, pp. 1405–1407, 2012.

[57] M Trentzsch, S Flachowsky, R Richter, J Paul, B Reimer, D Utess, S Jansen, H Mulaosmanovic, S Müller, S Slesazeck, *et al.*, "A 28nm hkmg super low power embedded nvm technology based on ferroelectric fets," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, IEEE, 2016, pp. 11–5.

[58] J. Muller, T. S. Boscke, U. Schroder, R. Hoffmann, T. Mikolajick, and L. Frey, "Nanosecond polarization switching and long retention in a novel mfis-fet based on ferroelectric fet," *IEEE Electron Device Letters*, vol. 33, no. 2, pp. 185–187, 2012.

[59] M. Lee, S.-T. Fan, C.-H. Tang, P.-G. Chen, Y.-C. Chou, H.-H. Chen, J.-Y. Kuo, M.-J. Xie, S.-N. Liu, M.-H. Liao, *et al.*, "Physical thickness 1. x nm ferroelectric hfzrox negative capacitance fets," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, IEEE, 2016, pp. 12–1.

[60] H Mulaosmanovic, J Ocker, S Muller, M Noack, J Mller, P Polakowski, T Mikolajick, and S Slesazeck, "Novel ferroelectric fet based synapse for neuromorphic systems," in *VLSI Technology, 2017 Symposium on*, IEEE, 2017, T176–T177.

[61] G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, *et al.*, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3498–3507, 2015.

[62] A. F. Vincent, J. Larroque, N. Locatelli, N. B. Romdhane, O. Bichler, C. Gamrat, W. S. Zhao, J.-O. Klein, S. Galdin-Retailleau, and D. Querlioz, "Spin-transfer

torque magnetic memory as a stochastic memristive synapse for neuromorphic systems," *IEEE transactions on biomedical circuits and systems*, vol. 9, no. 2, pp. 166–174, 2015.

[63] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2015, pp. 476–488.

[64] Y. Long, E. M. Jung, J. Kung, and S. Mukhopadhyay, "Reram crossbar based recurrent neural network for human activity detection," in *2016 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2016, pp. 939–946.

[65] X. Wu, V. Saxena, and K. Zhu, "A cmos spiking neuron for dense memristor-synapse connectivity for brain-inspired computing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2015, pp. 1–6.

[66] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, "A 3.1 mw 8b 1.2 gs/s single-channel asynchronous sar adc with alternate comparators for enhanced speed in 32 nm digital soi cmos," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 12, pp. 3049–3058, 2013.

[67] R Goldman, K Bartleson, T Wood, K Kranen, V Melikyan, and E Babayan, "32/28nm educational design kit: Capabilities, deployment and future," in *2013 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, IEEE, 2013, pp. 284–288.

[68] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm early design exploration," *IEEE Transactions on Electron Devices*, vol. 53, no. 11, pp. 2816–2823, 2006.

[69] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075 201, 2012.

[70] L. Gao, I.-T. Wang, P.-Y. Chen, S. Vrudhula, J.-s. Seo, Y. Cao, T.-H. Hou, and S. Yu, "Fully parallel write/read in resistive synaptic array for accelerating on-chip learning," *Nanotechnology*, vol. 26, no. 45, p. 455 204, 2015.

[71] M. F. Snoeij, A. J. Theuwissen, K. A. Makinwa, and J. H. Huijsing, "Multiple-ramp column-parallel adc architectures for cmos image sensors," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 12, pp. 2968–2977, 2007.

[72] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *Proceedings of the 53rd annual design automation conference*, ACM, 2016, p. 19.

[73] M. Price, J. Glass, and A. P. Chandrakasan, "14.4 a scalable speech recognizer with deep-neural-network acoustic models and voice-activated power gating," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, IEEE, 2017, pp. 244–245.

[74] *R. atienza. (2017). lstm by example using tensorflow*, `https://github.com/roatienza/Deep-Learning-Experiments`.

[75] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones.," in *Esann*, 2013.

[76] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[77] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H.-S. P. Wong, "A neuromorphic visual system using rram synaptic devices with sub-pj energy and tolerance to variability: Experimental characterization and large-scale modeling," in *Electron Devices Meeting (IEDM), 2012 IEEE International*, IEEE, 2012, pp. 10–4.

[78] S. Yu, Z. Li, P.-Y. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian, "Binary neural network with 16 mb rram macro chip for classification and online training," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, IEEE, 2016, pp. 16–2.

[79] C Nail, G Molas, P Blaise, G Piccolboni, B Sklenard, C Cagli, M Bernard, A Roule, M Azzaz, E Vianello, *et al.*, "Understanding rram endurance, retention and window margin trade-off using experimental results and simulations," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, IEEE, 2016, pp. 4–5.

[80] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075 201, 2012.

[81] S Park, A Sheri, J Kim, J Noh, J Jang, M Jeon, B Lee, B. Lee, B. Lee, and H Hwang, "Neuromorphic speech systems using advanced reram-based synapse," in *Electron Devices Meeting (IEDM), 2013 IEEE International*, IEEE, 2013, pp. 25–6.

[82] M Trentzsch, S Flachowsky, R Richter, J Paul, B Reimer, D Utess, S Jansen, H Mulaosmanovic, S Müller, S Slesazeck, *et al.*, "A 28nm hkmg super low power

embedded nvm technology based on ferroelectric fets," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, IEEE, 2016, pp. 11–5.

[83] J. Muller, T. S. Boscke, U. Schroder, R. Hoffmann, T. Mikolajick, and L. Frey, "Nanosecond polarization switching and long retention in a novel mfis-fet based on ferroelectric devices," *IEEE Electron Device Letters*, vol. 33, no. 2, pp. 185–187, 2012.

[84] S. F. Müller, *Development of HfO2-Based Ferroelectric Memories for Future CMOS Technology Nodes*. BoD–Books on Demand, 2016, vol. 5.

[85] M. Lee, S.-T. Fan, C.-H. Tang, P.-G. Chen, Y.-C. Chou, H.-H. Chen, J.-Y. Kuo, M.-J. Xie, S.-N. Liu, M.-H. Liao, *et al.*, "Physical thickness 1. x nm ferroelectric hfzrox negative capacitance fets," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, IEEE, 2016, pp. 12–1.

[86] J Müller, E Yurchuk, T Schlösser, J Paul, R Hoffmann, S Müller, D Martin, S Slesazeck, P Polakowski, J Sundqvist, *et al.*, "Ferroelectricity in hfo 2 enables nonvolatile data storage in 28 nm hkmg," in *VLSI Technology (VLSIT), 2012 Symposium on*, IEEE, 2012, pp. 25–26.

[87] T. Böscke, J Müller, D Bräuhaus, U Schröder, and U Böttger, "Ferroelectricity in hafnium oxide: Cmos compatible ferroelectric field effect transistors," in *Electron Devices Meeting (IEDM), 2011 IEEE International*, IEEE, 2011, pp. 24–5.

[88] T. Böscke, S. Teichert, D Bräuhaus, J Müller, U Schröder, U Böttger, and T Mikolajick, "Phase transitions in ferroelectric silicon doped hafnium oxide," *Applied Physics Letters*, vol. 99, no. 11, p. 112 904, 2011.

[89] M. Kobayashi, N. Ueyama, K. Jang, and T. Hiramoto, "Experimental study on polarization-limited operation speed of negative capacitance fet with ferroelectric hfo 2," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, IEEE, 2016, pp. 12–3.

[90] H Mulaosmanovic, J Ocker, S Müller, M Noack, J Müller, P Polakowski, T Mikolajick, and S Slesazeck, "Novel ferroelectric fet based synapse for neuromorphic systems," in *VLSI Technology, 2017 Symposium on*, IEEE, 2017, T176–T177.

[91] M. E. Lines and A. M. Glass, *Principles and applications of ferroelectrics and related materials*. Oxford university press, 1977.

[92] Y. Kaneko, Y. Nishitani, and M. Ueda, "Ferroelectric artificial synapses for recognition of a multishaded image," *IEEE Transactions on Electron Devices*, vol. 61, no. 8, pp. 2827–2833, 2014.

[93] H Mulaosmanovic, S Slesazeck, J Ocker, M Pesic, S Muller, S Flachowsky, J Müller, P Polakowski, J Paul, S Jansen, *et al.*, "Evidence of single domain switching in hafnium oxide based fefets: Enabler for multi-level fefet memory cells," in *Electron Devices Meeting (IEDM), 2015 IEEE International*, IEEE, 2015, pp. 26–8.

[94] J. Muller, T. S. Boscke, U. Schroder, R. Hoffmann, T. Mikolajick, and L. Frey, "Nanosecond polarization switching and long retention in a novel mfis-fet based on ferroelectric $hboxHfO2$," *IEEE Electron Device Letters*, vol. 33, no. 2, pp. 185–187, 2012.

[95] S. Oh, T. Kim, M. Kwak, J. Song, J. Woo, S. Jeon, I. K. Yoo, and H. Hwang, "Hfzro x-based ferroelectric synapse device with 32 levels of conductance states for neuromorphic applications," *IEEE Electron Device Letters*, vol. 38, no. 6, pp. 732–735, 2017.

[96] S. K. Samal, S. Khandelwal, A. I. Khan, S. Salahuddin, C. Hu, and S. K. Lim, "Full chip power benefits with negative capacitance fets," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2017, pp. 1–6.

[97] Y Nishitani, Y Kaneko, M Ueda, T Morie, and E Fujii, "Three-terminal ferroelectric synapse device with concurrent learning function for artificial neural networks," *Journal of Applied Physics*, vol. 111, no. 12, p. 124 108, 2012.

[98] H Mulaosmanovic, J Ocker, S Müller, M Noack, J Müller, P Polakowski, T Mikolajick, and S Slesazeck, "Novel ferroelectric fet based synapse for neuromorphic systems," in *VLSI Technology, 2017 Symposium on*, IEEE, 2017, T176–T177.

[99] Z. Wang, S. Khandelwal, and A. I. Khan, "Ferroelectric oscillators and their coupled networks," *IEEE Electron Device Letters*, vol. 38, no. 11, pp. 1614–1617, 2017.

[100] H. Ishiwara, "Proposal of adaptive-learning neuron circuits with ferroelectric analog-memory weights," *Japanese journal of applied physics*, vol. 32, no. 1S, p. 442, 1993.

[101] X. Li, S. George, K. Ma, W.-Y. Tsai, A. Aziz, J. Sampson, S. K. Gupta, M.-F. Chang, Y. Liu, S. Datta, *et al.*, "Advancing nonvolatile computing with nonvolatile ncfet latches and flip-flops," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2017.

[102] H. M. C. Consortium *et al.*, "Hybrid memory cube specification 1.0," *Last Revision Jan*, 2013.

[103] D. Williamson, "Dynamically scaled fixed point arithmetic," in *Communications, Computers and Signal Processing, 1991., IEEE Pacific Rim Conference on*, IEEE, 1991, pp. 315–318.

[104] T. Na and S. Mukhopadhyay, "Speeding up convolutional neural network training with dynamic precision scaling and flexible multiplier-accumulator," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ACM, 2016, pp. 58–63.

[105] T. Na, J. H. Ko, J. Kung, and S. Mukhopadhyay, "On-chip training of recurrent neural networks with limited numerical precision," in *Neural Networks (IJCNN), 2017 International Joint Conference on*, IEEE, 2017, pp. 3716–3723.

[106] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, ACM, 2014, pp. 675–678.

[107] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[108] B. Gao, Y. Bi, H.-Y. Chen, R. Liu, P. Huang, B. Chen, L. Liu, X. Liu, S. Yu, H.-S. P. Wong, *et al.*, "Ultra-low-energy three-dimensional oxide-based electronic synapses for implementation of robust high-accuracy neuromorphic computation systems," *ACS nano*, vol. 8, no. 7, pp. 6998–7004, 2014.

[109] H. Mulaosmanovic, S. Slesazeck, J. Ocker, M. Pesic, S. Muller, S. Flachowsky, J. Müller, P. Polakowski, J. Paul, S. Jansen, S. Kolodinski, C. Richter, S. Piontek, T. Schenk, A. Kersch, C. Kunneth, R. van Bentum, U. Schroder, and T. Mikolajick, "Evidence of single domain switching in hafnium oxide based fefets: Enabler for multi-level fefet memory cells," in *2015 IEEE International Electron Devices Meeting (IEDM)*, 2015, pp. 26.8.1–26.8.3.

[110] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, IEEE Press, 2018, pp. 764–775.

[111] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, IEEE Press, 2018, pp. 383–396.

[112] C.-C. Chou, Z.-J. Lin, P.-L. Tseng, C.-F. Li, C.-Y. Chang, W.-C. Chen, Y.-D. Chih, and T.-Y. J. Chang, "An n40 256k× 44 embedded rram macro with sl-precharge sa and low-voltage current limiter to improve read and write performance," in *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, IEEE, 2018, pp. 478–480.

[113] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[114] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[115] N. Silberman and S. Guadarrama, "Tensorflow-slim image classification model library," 2016.

[116] C. Baskin, E. Schwartz, E. Zheltonozhskii, N. Liss, R. Giryes, A. M. Bronstein, and A. Mendelson, "Uniq: Uniform noise injection for non-uniform quantization of neural networks," *arXiv preprint arXiv:1804.10969*, 2018.

[117] J. Choi, S. Venkataramani, V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and efficient 2-bit quantized neural networks," in *SysML 2019*, 2019.

[118] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[119] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[120] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[121] D. Zuras, M. Cowlishaw, A. Aiken, M. Applegate, D. Bailey, S. Bass, D. Bhandarkar, M. Bhat, D. Bindel, S. Boldo, *et al.*, "Ieee standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, 2008.

[122] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An instruction set architecture for neural networks," in *ACM SIGARCH Computer Architecture News*, IEEE Press, vol. 44, 2016, pp. 393–405.

[123] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2018, pp. 764–775.

[124] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International workshop on ambient assisted living*, Springer, 2012, pp. 216–223.

[125] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[126] *Google tpu-v2 analyses*, `https://www.nextplatform.com/2018/05/10/tearing-apart-googles-tpu-3-0-ai-coprocessor/`, Accessed: 2010-09-30.

[127] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[128] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "Wrpn: Wide reduced-precision networks," *arXiv preprint arXiv:1709.01134*, 2017.

[129] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 2019, pp. 925–938.

[130] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17, Toronto, ON, Canada: ACM, 2017, pp. 1–12, ISBN: 978-1-4503-4892-8.

[131] Y. Long *et al.*, "A Ferroelectric FET based Power-efficient Architecture for Data-intensive Computing," in *International Conference on Computer-Aided Design (ICCAD)*, 2018.

[132] W.-H. Chen *et al.*, "A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, IEEE, 2018, pp. 494–496.

[133] M.-Y. Lin *et al.*, "DL-RSIM: A Simulation Framework to Enable Reliable ReRAM-based Accelerators for Deep Learning," in *International Conference on Computer-Aided Design (ICCAD)*, 2018.

[134] T. Na *et al.*, "Speeding up convolutional neural network training with dynamic precision scaling and flexible multiplier-accumulator," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ACM, 2016, pp. 58–63.

[135] H. Li *et al.*, "Variation-aware, reliability-emphasized design and optimization of rram using spice model," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, IEEE, 2015, pp. 1425–1430.

[136] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

VITA

Yun Long received the B.S. degree in microelectronics from Peking University, Beijing, China in 2014. He is currently pursuing the Ph.D degree in Electrical and Computer Engineering with the Georgia Institute of Technology, Atlanta, GA, USA. His research insterests include emerging technology based machine learning accelerator design, process-in-memory architecture design, DNN model reduction techniques, and machine learning based dynamical system model. He received the Wusi Fellowship and National Fellowship in 2008 and 2009 when he was an undergraduate student in Peking University. Yun Long will join Google TPU team as as research scientist in Fall 2019.